UNIVERSITÉ PARIS-EST
LABORATOIRE D'INFORMATIQUE
ÉCOLE DOCTORALE MSTIC

THÈSE
Pour l'obtention du titre de
DOCTEUR DE L'UNIVERSITÉ PARIS-EST
Spécialité Informatique

Modélisation et rendu temps-réel de milieux participants à l'aide du GPU

MODELING AND REAL-TIME RENDERING OF PARTICIPATING MEDIA USING THE GPU

Soutenue le 18 décembre 2012 par

Anthony Giroud

Dirigée par
Venceslas Biri

Composition du jury :

Rapporteur :	Jean-Michel DISCHLER	LSIIT - Université de Strasbourg
Rapporteur :	Christophe RENAUD	LISIC - Université du Littoral - Côte d'Opale
Examinateur :	Yannick RÉMION	CReSTIC - Université de Reims - Champagne-Ardenne
Examinateur :	Jean-Philippe FARRUGIA	LIRIS - Université Lyon 1
Examinateur :	Caroline CHAUX	LATP / CNRS - Université Aix-Marseille

Résumé

Cette thèse traite de la modélisation, l'illumination et le rendu temps-réel de milieux participants à l'aide du GPU.

Dans une première partie, nous commençons par développer une méthode de rendu de nappes de brouillard hétérogènes pour des scènes en extérieur. Le brouillard est modélisé horizontalement dans une base 2D de fonctions de Haar ou de fonctions B-Spline linéaires ou quadratiques, dont les coefficients peuvent être chargés depuis une *fogmap*, soit une carte de densité en niveaux de gris. Afin de donner au brouillard son épaisseur verticale, celui-ci est doté d'un coefficient d'atténuation en fonction de l'altitude, utilisé pour paramétrer la rapidité avec laquelle la densité diminue avec la distance au milieu selon l'axe Y.

Afin de préparer le rendu temps-réel, nous appliquons une transformée en ondelettes sur la carte de densité du brouillard, afin d'en extraire une approximation grossière (base de fonctions B-Spline) et une série de couches de détails (bases d'ondelettes B-Spline), classés par fréquence. Chacune de ces bases de fonctions 2D s'apparente à une grille de coefficients.

Lors du rendu sur GPU, chacune de ces grilles est traversée pas à pas, case par case, depuis l'observateur jusqu'à la plus proche surface solide. Grâce à notre séparation des différentes fréquences de détails lors des pré-calculs, nous pouvons optimiser le rendu en ne visualisant que les détails les plus contributifs visuellement en avortant notre parcours de grille à une distance variable selon la fréquence. Nous présentons ensuite d'autres travaux concernant ce même type de brouillard : l'utilisation de la transformée en ondelettes pour représenter sa densité via une grille non-uniforme, la génération automatique de cartes de densité et son animation à base de fractales, et enfin un début d'illumination temps-réel du brouillard en simple diffusion.

Dans une seconde partie, nous nous intéressons à la modélisation, l'illumination en simple diffusion et au rendu temps-réel de fumée (sans simulation physique) sur GPU. Notre méthode s'inspire des Light Propagation Volumes (volume de propagation de lumière), une technique à l'origine uniquement destinée à la propagation de la lumière indirecte de manière complètement diffuse, après un premier rebond sur la géométrie.

Nous l'adaptons pour l'éclairage direct, et l'illumination des surfaces et milieux participants en simple diffusion. Le milieu est fourni sous forme d'un ensemble de bases radiales (blobs), puis est transformé en un ensemble de voxels, ainsi que les surfaces solides, de manière à disposer d'une représentation commune. Par analogie aux LPV, nous introduisons un Occlusion Propagation Volume, dont nous nous servons, pour calculer l'intégrale de la densité optique entre chaque source et chaque autre cellule contenant soit un voxel du milieu, soit un voxel issu d'une surface. Cette étape est intégrée à la boucle de rendu, ce qui permet d'animer le milieu participant ainsi que les sources de lumière sans contrainte particulière. Nous simulons tous types d'ombres : dues au milieu ou aux surfaces, projetées sur le milieu ou les surfaces.

Mots-clés : milieu participant, illumination, brouillard, fumée, GPU, ondelettes

Abstract

This thesis deals with modeling, illuminating and rendering participating media in real-time using graphics hardware.

In a first part, we begin by developing a method to render heterogeneous layers of fog for outdoor scenes. The medium is modeled horizontally using a 2D Haar or linear/quadratic B-Spline function basis, whose coefficients can be loaded from a fogmap, i.e. a grayscale density image. In order to give to the fog its vertical thickness, it is provided with a coefficient parameterizing the extinction of the density when the altitude to the fog increases.

To prepare the rendering step, we apply a wavelet transform on the fog's density map, and extract a coarse approximation and a series of layers of details (B-Spline wavelet bases). These details are ordered according to their frequency and, when summed back together, can reconstitute the original density map.

Each of these 2D function basis can be viewed as a grid of coefficients. At the rendering step on the GPU, each of these grids is traversed step by step, cell by cell, since the viewer's position to the nearest solid surface. Thanks to our separation of the different frequencies of details at the precomputations step, we can optimize the rendering by only visualizing details that contribute most to the final image and abort our grid traversal at a distance depending on the grid's frequency.

We then present other works dealing with the same type of fog: the use of the wavelet transform to represent the fog's density in a non-uniform grid, the automatic generation of density maps and their animation based on Julia fractals, and finally a beginning of single-scattering illumination of the fog, where we are able to simulate shadows by the medium and the geometry.

In a second time, we deal with modeling, illuminating and rendering full 3D single-scattering sampled media such as smoke (without physical simulation) on the GPU. Our method is inspired by light propagation volumes, a technique whose only purpose was, at the beginning, to propagate fully diffuse indirect lighting. We adapt it to direct lighting, and the illumination of both surfaces and participating media. The medium is provided under the form of a set of radial bases (blobs), and is then transformed into a set of voxels, together with solid surfaces, so that both entities can be manipulated more easily under a common form.

By analogy to the LPV, we introduce an occlusion propagation volume, which we use to compute the integral of the optical density, between each source and each other cell containing a voxel either generated from the medium, or from a surface. This step is integrated into the rendering process, which allows to animate participating media and light sources without any further constraint.

Keywords : participating medium, illumination, fog, smoke, GPU, wavelets

Remerciements / Acknowledgements

Je souhaite avant tout remercier les membres de mon jury de thèse. Un grand merci aux deux rapporteurs, Jean-Michel Dischler, du LSIIT de l'Université de Strasbourg, et Christophe Renaud, du LISIC de l'Université Littoral - Côte d'Opale, pour avoir fait l'effort de relire mon manuscrit. Je remercie aussi les examinateurs, Jean-Philippe Farrugia du LIRIS de l'Université Lyon 1, Yannick Rémion, du CReSTIC de l'Université de Reims Champagne-Ardenne, et enfin Caroline CHAUX du LATP/CNRS de l'Université Aix-Marseille pour avoir accepté de siéger dans le jury et sans qui la soutenance n'aurait pas été possible.

Je souhaite remercier mon directeur de thèse, Venceslas Biri, pour m'avoir supporté durant plus de sept longues années. Entre 2005 et 2008, il m'enseigna la Synthèse d'Images durant mon cursus à l'école d'ingénieurs IMAC, où il fit preuve de beaucoup de pédagogie. Je le remercie infiniment de m'avoir accepté dans son équipe pour mon stage de fin d'études, et d'avoir ensuite fait le pari de m'accepter en thèse. Durant ces quatre années, il fut d'un grand conseil, fut toujours disponible, et n'a jamais cessé de me soutenir.

Je remercie Gilles Bertrand pour m'avoir accueilli au sein du laboratoire A3SI dont il est le directeur. Il a eut la gentillesse d'être mon directeur de thèse officiel dans les premiers temps.

Je souhaite également remercier les autres personnes que j'ai eu la chance de côtoyer quotidiennement durant ces quelques années. Un grand merci à Vincent Nozick, qui a également été très présent, m'a fourni de nombreux conseils pratiques et n'a eu de cesse de m'encourager. Je remercie mes collègues de bureau, autres doctorants, qui ont contribué à donner au laboratoire son ambiance chaleureuse : Benjamin Raynal, Fabrice Boutarel, Nadine Dommanget, Adrien Herubel, Christophe Guentleur, Imen Melki, François de Sorbier, Patrice Bouvier, Pierre Boulenguez, John Chaussard, Yohan Thibault, Camille Couprie, Phuc Ngo, Dalila Trad, Ravi Kiran, et tous ceux que j'aurais involontairement oubliés. Je remercie aussi Michel Couprie, Yukiko Kenmochi, Hugues Talbot, Jean Cousty et les autres professeurs et membres de l'équipe A3SI.

Je n'oublie pas certaines personnes qui ont un rapport plus secondaire avec ma thèse, mais qui ont néammoins joué un rôle crucial. Merci à Sylvie Donard, secrétaire de formation à l'IMAC, pour sa gentillesse et à qui je dois toute la planification et l'organisation de mes enseignements. Merci également à Sylvie Cach, responsable administrative l'école doctorale MSTIC, qui m'a fourni toute l'aide logistique dont j'ai eu besoin pour l'organisation de ma soutenance.

En dernier mais pas des moindres, un grand merci à mes parents et à ma soeur. J'ai la chance d'avoir une famille qui a toujours cru en moi, et n'a jamais ménagé ses efforts pour me soutenir durant tout mon parcours, que ce soit moralement ou de façon plus pratique. Enfin, je remercie ma femme Qing, qui m'a donné toute la motivation nécessaire pour aller jusqu'au bout de cette thèse.

To Qing,
To my family...

Contents

List of figures

Chapter 1

Introduction

"It's not put into his head to be buried. It's put into his head to be made useful. You hold your life on the condition that to the last you shall struggle hard for it. Every man holds a discovery on the same terms."

Charles Dickens, *Little Dorrit*

1.1 From line drawing to participating media rendering

1.1.1 1960s : drawing lines and filling polygons

In the field of digital imagery, rendering is the process by which a virtual scene is visualized. The content of the scene, a set of mostly two-dimensional objects, is described by a mathematical model providing a description of all objects' shape and appearance, as well as other elements contributing to their visual appearance, such as light sources, participating media, etc. A virtual camera is configured with a given projection, assigned a particular position within the scene, and the image seen from its point of view is calculated by the computer. The whole rendering process can be real-time ($\sim$24-25 images per second) as well as it can last several hours, depending on the rendering algorithm, the computer's hardware capabilities and the complexity of the scene.

In 1963, Ivan Edward Sutherland presented Sketchpad, the first computer-aided drafting system. This work was part of its PhD thesis at the Massachusetts Institute of Technology (M.I.T.), and later gave him the Turing Award in 1988. Developed exclusively for the M.I.T.'s Lincoln TX-2 computers, the program introduced many concepts used in today's object programming. Using a light pen, the user could directly shape simple objects on the 2D point plotter screen (lines, squares, circles, etc.). It was possible to create several instances of a father object, as well as applying constraints on angles and distances. The Sketchpad system is considered by many as one of the first systems showing a true direct man-machine interaction and featuring a complete user interface. A lot of researchers also consider the introduction of this system as the date when computer graphics truly took off.

With the need to output both text and simple 2D shapes, raster graphics started to develop. Unlike vector monitors, where a beam traced and refreshed repeatedly simple lines on the screen's surface, and thus could only display unfilled basic shapes outlines, raster monitors displayed information under the form of a discrete grid, containing a set of squares (i.e. picture elements, or pixels) which are turned on and off to approximate the image. Raster display monitors were already used, such as the classical CRT screens featured in televisions. Although TV images were only discretized vertically (e.g. line-by-line refreshment, horizonal resolution could slightly change), computer generated raster graphics had to be fixedly discretized along the two dimensions. Raster graphics made it possible to display coloured filled shapes, and their discrete aspect induced new problematics, such as aliasing. Vector-based screens were to continue to be used for some time, for example in the Vectrex video game system, manufactured by G.C.E. in 1982.

In his work entitled *Some techniques for shading machine renderings of solids*, published in 1968, Arthur Appel evocates how spacial considerations like not displaying hidden surfaces and shadow casting objects situated behind others could improve visual and spatial realism, as well as emphasizing particular parts of the drawing. He introduced several concepts which were to be developed years later with 3D rendering algorithms such as the ray-tracing.

1.1.2 1970s : making polygons look realistic

The problem of not displaying elements situated behind others from the camera's point of view had already been raised several years before, first by Appel concerning hidden lines, and then by Warnock and Watkins for surfaces, and several solutions had been proposed. In the early 1970s, the state of the art solution to hidden surfaces removal was the so-called painter's algorithm. The idea was to initially sort surfaces according to their distance from the camera's point of view, so that by drawing them from the farthest to the nearest, hidden surfaces find themselves naturally overlayed by the next object just in front of them. The major drawback of the painter's algorithm is its failure to handle ambiguous cases, when it is impossible to clearly sort surfaces, for example with three surfaces where each one is partly situated both under a second surface and over a third one.

In 1974, Edwin Catmull presented the z-buffering technique, an efficient solution to the problem of hidden surfaces removal in the context of raster graphics. When the first surface in front of the camera is rasterized, the depth of each fragment is stored on a buffer, besides the classical frame buffer storing its color. When another surface has some of its fragments overlaying those of the previous surface, their two respective depths are compared, and only the fragment which depth is the lowest is kept. This process is then repeated with all other surfaces. Z-buffering is still the method currently implemented on graphics hardware, but is becoming increasingly costly in terms of memory bandwidth, due to the increasing resolution of images.

Shading an object consists in determining its visual appearance, mainly in order to simulate the effects of lighting on surfaces. Before the first smooth shading models emerged, people used flat shading, a low-cost model which only accounts for the angle between the surface's normal vector and the lighting direction, the intensity of the source and respective colors of both the surface and the source. Although very fast, it provides low quality results where the polygons are clearly apparent.

In 1971, Henri Gouraud introduces his shading model, a simple modification of flat shading where normal vectors are associated to each individual vertices as the average normal of all neigh-

bouring surfaces linked to this vertex. The model also additionally accounts for the quadratic attenuation caused by the distance between each vertex and each light source. Gouraud shading provides clearly better visual results in comparison to flat shaded scenes, with smooth continuous light reflection changes at polygon intersections.

Bui Thong Phong's shading model was published in 1973 as part of his PhD thesis. Gouraud's model which computes the illumination on each vertex, then interpolates radiances between all vertices delimiting a surface to obtain the color at each point. Phong's model directly evaluates the emitted radiance on each point of the surface, this is why it was considered as far too heavy at the time. By comparison to Gouraud shading, Phong also inserts a specular term in the equation, together with the diffuse term.

The concept of texture mapping was also introduced by Edwin Catmull in 1974, as part of its doctoral thesis. An image, the texture, which can theoretically be nD is laid onto an object's surface, this way quickly and greatly adding realism in an even very simple scene, while inducing only a low additional cost. Texture mapping is implemented in today's graphics hardware and is now a compulsory feature of modern modeling software.

1.1.3 From 1980s : the golden age of computer graphics

1.1.3.1 Two milestone rendering algorithms

Since the late 1960s, researchers had been trying in vain to find a way to generate virtual images with a degree of photorealism approaching real-life pictures.

In 1980, John Turner Whitted published an edifying article entitled *An improved illumination model for shaded display* [Whi80], in which he presented the ray-tracing, an algorithm making it possible to obtain both reflection and refraction of light on a surface, as well as shadows. Its main principle is to reproduce the behavior of light rays, but in the reverse direction, from the camera towards the scene.

While being very simple to understand, it only required a moderate amount of code for being implemented. Although its major drawback is its huge computational cost, not really making it the algorithm of choice for real-time rendering applications, the visual results it provided were astonishing at the time.

The radiosity illumination method was presented in 1984 by Allen C. Goral [GTG84].

The algorithm starts with the division of all surfaces of the scene into small parts, called *patches*. In a second step, the direct visibility from each single patch of all other patches is calculated, taking into account the distance, the angle and possible occluding surfaces between the two patches, resulting in a value, called the *form factor*. All form factors are then inserted in a set of linear equations, expressing the amount of energy emitted by each individual patch, depending on the energy of all other patches. The third step consists in iteratively transferring energy between each pair of patches, before finally rendering the patches to visualize the result.

Although raytracing is a local illumination method and therefore can only generate hard shadows caused by direct lighting, radiosity computes global illumination involving multiple bounces of light across the scene.

1.1.3.2 Firsts graphics cards and the OpenGL library

In the early 1980s, several computer manufacturers started to equip their systems with a coprocessor (i.e. *blitter*) dedicated to operations related to rendering, such as rasterization. In comparison to today's graphics cards, blitters were limited to very simple logical operations on binary pixel data. GPUs have the ability to perform a much wider range of mathematical and geometrical operations, with dedicated hardware functionalities for most of them, such as matrix operations, z-buffering, rasterization, values interpolation between vertices, shaders, etc.

The first devices resembling a video card were IBM's MDA (Monochrome Display Adapter) and CDA (Color Display adapter) cards, launched in 1981. They were designed to equip the IBM PC, and came as separate cards installed on the motherboard, and on which the monitor was directly plugged. In 1987, IBM also launched the VGA (Video Graphics Array) card, featuring a new display standard, the VGA, which was soon widely adopted by a wide range of computer display manufacturers.

Modern GPUs appeared in 1995, with the NVIDIA NV1 and 3DFX JAMMA GCI, which brought 3D hardware acceleration to PCs. At this time, not all 3D chips used polygons as primitives. It is not until 1997 that GPUs will start to widely implement OpenGL and Direct3D pipelines.

Open Graphics Library was launched in 1992 by Silicon Graphics (SGI), and soon became the standard library for real-time 3D graphics. Before OpenGL, some professional computer graphics systems such as those of SGI were already featuring 3D hardware acceleration for SGI's IrisGL library since the mid-1980s, but consumer-grade PCs were not. Built on IrisGL, the main purpose of the OpenGL library was both to provide a common interface for graphics programmers, and to simplify the implementation of computer graphics programs by handling all communications with the hardware. If the system is equipped with graphical hardware acceleration, OpenGL will automatically exploit these features, otherwise it will switch to software emulation, without necessiting any particular action from the programmer.

1.2 The challenge of rendering participating media

At the beginning of the years 1980, there was a true interest by researchers in trying to render increasingly realistic scenes. Most techniques to simulate convincing surfaces were known, using models like [CT82], by mapping textures or perturbating normals. The newly published raytracing [Whi80] enabled the simulation of much more sophisticated optical effects like mirror reflections and refractions, making it possible to model the interactions of light with objects' surfaces very faithfully, by attempting to reproducing the physical phenomena governing geometrical optics.

One of the problems of raytracing, is that the scenes look, in a sense, too artificial, due to the absence of diffuse reflections and soft shadows, obtained with a global illumination algorithm. This was only one part of the problem, something else was also lacking in order for computer generated images to really look photorealistic : non-surfacic objects such as fog, smoke, etc. Participating media can be found everywhere in nature, for the reason that light does not simply travel in a vacuum between surfaces. In indoor scenes, where the visibility distance is very limited due to occlusions from the walls, an efficient global illumination model was enough to provide good quality results, but a number of beautiful phenomena could be rendered by additionally simulating light that scatters through dust floating in the air. In outdoor scenes, participating media became

absolutely unavoidable, to simulate volumic meteorological phenomena like clouds or fog, or the reduction of the visibility due to ambient humidity when the observer looks at the horizon.

In comparison with surfaces, rendering participating media continues to pose a true challenge to researchers. Contrarily to surfaces which are defined very easily using a few vertices and some material properties, participating media are, in essence, volumetric entities, which adds another dimension to the problem. With raytracing, for example, when a view-ray strikes a surface, it stops there and cannot undergo further events. In case that the surface is translucent, interactions with other surfaces lying behind are possible, but only through a separate ray, possibly emitted in the same direction from the other side of the surface, after a refraction.

In the presence of a participating medium, interactions happen in a continuous manner. Although gases and vapor are composed of microscopic particles, they cannot be rendered the same way as surfaces, since, due to their small size, each single interaction with a light ray is not sufficient to block it completely. Instead, only a small amount of light is deflected in other directions, while the rest continues its way straight forward. Even the simplest situation, without any illumination, whereas visualizing a polygon only involves colouring each pixel with the right polygon's color, rendering a medium supposes to integrate its density along the view ray. This is only straightforward for the case of homogeneous or analytical media, which are, unfortunately, the less convincing models, barely encountered in the nature.

As we shall see in part two of this manuscript, there are three main ways of representing a medium in computer graphics. The first is based on an analytical definition of the distribution of its density in the scene. When we have a mathematical formula explicitely associating a 3D position with the corresponding density, any integration between two positions can be achieved mathematically and provide an exact result. The second is using a set of spherical particles called blobs, where the density is at its maximum at each particle's center, and decreases exponentially as we move away. The last one employs a regular 3D grid of voxels in which the density is discretely sampled. Each model has its advantages and drawbacks.

All these issues can be summarized in the following questions :

- How could we efficiently model and represent fog in memory ?
- Instead of the binary classification of media as homogeneous or heterogeneous, is there another approach possible in which smoother heterogeneous media would be faster to render than more detailed media of the same resolution ?
- Is an analytical animation of fog possible ?
- Can the GPU be used to integrate optical depth around light sources, without rendering the scene from their positions ?
- Can we take into account occlusions by the medium and by the geometry using a coherent framework ?

1.3 Specifications and contraints for our approach

We answer the problematics above by proposing two distinct participating media rendering methods.

We considered beginning with a work on the visualization of simple media like outdoor fog, and gave ourselves the following constraints :

- Our method must be real-time, with at least $25 \sim 30$ images per second on conventional mainstream graphics cards.

- We want to take advantage of the smoothness of such media to speed-up the rendering. Multiresolution analysis seems the way to go, and, more specifically, the sophisticated mathematical tool of wavelet analysis looks promising for its accuracy and optimized representation of data. Wavelets find many applications in domains such as data compression.

- Our fog must be heterogeneous, in the sense that the density can be modifiable locally. This is only possible using particles like blobs, or a grid of samples.

- Fog is a very smooth phenomenon, and we rarely observe sharp contrasts in the density. We would like to ensure this smoothness in the visual result, even if the density distribution, as provided by the user, features such unnatural artefacts. One solution is to multiply the samples by a smooth function, such as a Gaussian or a B-Spline.

- Seen from a distant point of view, fog looks more like a horizontal sheet rather than a cube like clouds, or vertical column such as smoke. We cannot avoid to take this into account, therefore we choose to model our fog using a horizontal two-dimensional function basis along the horizontal XZ plane, and then give the medium its vertical thickness using a vertical density extinction coefficient. Since we plan to use wavelet analysis, our medium must be modeled in a scaling function basis.

- Make it very easy and convenient to design the appearance of the medium. In our work, the user can import a grayscale image to simply distribute the density across the medium.

- As we highlight in our state of the art, true heterogeneity in the medium does not really get on well with analytical solutions, when the density is parameterizable locally using density samples. However, we want to avoid a basic regular-step numerical integration at runtime, and find a "locally analytical", more accurate manner of integrating over the fog's density. We address this by performing a smart grid traversal, where we actually compute the intersections of the view-ray with the grid and advance cell-by-cell, combined with an accurate precomputation of integrals across the domain of each basis functions.

Later in this thesis, we worked on a new method to illuminate mediums such as smoke. This is a very active area of research, with lots of good publications in the past five years.

We wanted our solution to fulfill the following specifications :

- It must run in real-time on general public GPUs.

- We want to take the geometry of the scene into account, as a lot of applications need not only to visualize volumes, but often integrate it in a scene featuring surfacic objects. Our strategy is based on the voxelization of both the medium and the geometry inside a common density volume.

- We want our solution to be fully deterministic and predictable, in order to avoid the common artefacts of Monte-Carlo solutions. Our approach is based on an iterative propagation of light inside a volume. Instead of considering light as a set of individual rays or photons, our solution simulates light more as a wavefront which propagates from the sources throughout the scene. Such an approach is easily implementable on any programmable GPU using a couple of shaders.

- The medium must be easily modelable, while offering a true 3D heterogeneity. As we shall see in chapters 6 and 7, the two main ways of representing heterogeneous participating media are either a set of particles, or a regular 3D grid. We choose to combine both of them, by allowing the user to model the medium using such particles, while our application mainly manipulates volumetric grids.

- We wanted our solution to handle several light sources. In our implementation, we allow up to eight point light sources.

- The lighting conditions (position of sources) and the participating medium must be dynamic. We only precompute the geometry voxelization step.

1.4 Outline of this manuscript

This manuscript is divided in three parts, each organized as follows.

The first part is dedicated to providing some theoretical prerequisites that seem necessary to understand in depth the issues addressed in the state of the art as well as in our own work.

- In chapter 2, we begin with a brief introduction to the physics of light in the real world, to better understand certain concepts such as a beam of light, a photon, as well as phenomena such as reflection, or the relationship between color and the wavelength.

- In chapter 3, we characterize mathematically the possible interactions of light with the particles in a medium, and assemble them together to form the transport equation, which formally summarizes the energy entering and exiting an unit volume in a medium. We then introduce the notion of radiance, and finally establish the equation of transfer, which is probably the central equation in participating media illumination.

- In chapter 4, we quickly introduce the theory behind wavelet basis functions and the wavelet decomposition algorithm. We use wavelets in our work on fog rendering, to efficiently separate the different frequencies of details from the medium's density distribution.

In the second part, we review existing techniques in the literature of participating media visualization.

- In chapter 5, we make a brief foreword to our state of the art. We present the usual distinction between analytical, deterministic and stochastic rendering techniques, and explain why it is justified.

- In chapter 6, we begin with some of the most notable methods among ancient works, before real-time became achievable in volume illumination, like traditional solid geometry. We compare the different strategies for modeling participating media depending on their type.

- In chapter 7, we discuss real-time rendering techniques using graphics hardware and consider three types of media : fog, clouds and smoke. These various techniques help showing the difference between programmable and non-programmable GPUs, where the rendering techniques mainly relied on features like texture mapping and hardware blending.

In the third and final part, we present our contributions.

- In chapter 8, we first present a simple method for fast rendering of non-illuminated heterogeneous fog based on a B-Spline wavelet decomposition. This work was published at the **WSCG'2010 international conference** in Plzeň, Czech Republic, and led to a national publication at the AFIG'2009 conference.

- In chapter 9, we then present our attempts to further speed-up the visualization of this kind of fog based on a quad-tree and the separate rendering of each coefficient of the wavelet basis modeling the medium's density distribution. Unfortunately, this research was put on hold and is not published at this time.

- In chapter 10, we continue with a simple method to design such fog using Perlin noise and render it in real-time, which was presented in the **Chaos'2010 international conference**.

- In chapter 11, we finish this part with our latest work where we consider the single-scattering illumination of our fog by the sun. Since this technique is still in development, it also did not let to a publication yet.

- In chapter 12, we present a completely different method to illuminate a single-scattering medium by several point light sources. We simulate effects such as glows around sources positioned inside the medium, as well as shadows by both the medium and the geometry. This method is implemented on the GPU and is based on an iterative propagation of the optical depth around light sources throughout a regular 3D grid. This work was published at **GRAPP'2011 international conference** in Algarve, Portugal.

Part I

Theoretical background

Chapter 2

Light and its properties

"There are mysteries which men can only guess at, which age by age they may solve only in part."

Bram Stocker, *Dracula*

2.1 A brief history

2.1.1 Early theories

Explaining light has never been a simple and natural question.

Before the physical nature of light, for centuries, the first big question that came to early physicists and philosophers concerned its speed. Early theoricians and physicists had a different notion of light sources and the spatial origin of light when they were looking at objects that they could see. The first written track of someone questioning the nature of light was Empedocles of Acragas (492-432 BC), a greek philosopher, who formulated the first notable theory of vision.

He stated that we can see objects in front of us because particles emitted from our eyes travel in the air and eventually make contact with objects in front of us (figure 2.1). Later, greek mathematicians and philosophers such as Euclid ($\sim 300BC$) will try to form a more comprehensive theory of light, based on this view.

Aristotle (384-322BC) in *De Sensu*, disagreed with Empidocles' view of light as an entity travelling from one place to another, thus necessarily performing a movement. He argued that this vision contraries all observable facts, and writes that light might well be due to the presence of something, but that something is not in movement. If light does not move, it has no finite speed.

Centuries later, the Greeks still believed that light was emitted by their eyes. When looking at the nightsky, Hero of Alexandria (10 - 70 AD) came to the conclusion that light had to be able to travel back and forth to very distant objects instantly, so that he could see stars immediately when looking at them.

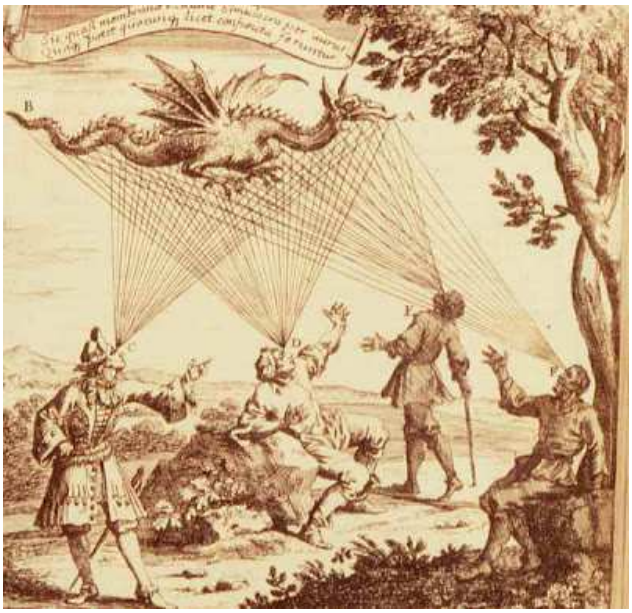

Figure 2.1: Medieval illustration depicting the ancient Greek theory of vision : light rays are emitted from the eye and go fill the object.

2.1.2 Roemer and the proof of the finite speed of light

Due to the lack of adapted observation tools, this question of the speed of light remained unanswered until 1676, when Danish astronomer Ole Roemer (1644 - 1710) came with the first real proof that light could not travel instantly between two distant points.

Other notable astronomers like René Descartes (1596 - 1650) or Galileo Galilei (1564 - 1642) had conducted experiments to try measuring the speed of light, but eventually failed because their measurements were conducted between to points distant of only a few kilometers, which was far too short to humanely notice any discrepancy between the instant when light was emitted from one point and the instant it was received at the other point.

Contemporary astronomers already knew that orbiting objects in space keep moving at a constant speed. However, when studying Jupiter's closest satellite Io and trying to measure how long it takes to complete a revolution, Roemer noticed that he obtained results which seemed to vary exactly according to the distance between Earth and Io.

Once the first proof of a finite speed of light established, Roemer also came with a first estimate of the speed of light : $\simeq 225,300,000$ meters per second (today estimated at 299,792,458 m/s), which is a brillant result, considering the tools and approximative astronomical measures available at the time.

2.1.3 Diffraction : how are colors generated ?

2.1.3.1 Light and dark theory

Before the 1660s and the beginning of modern experiments on light, people though that all colors could be generated by basically combining both light and darkness in different proportions. In

Figure 2.2: Sir Isaac Newton (1642 - 1726)

Optics in Six Chapters, published in 1613, François d'Aguilon (1567 - 1617), a belgian mathematician, gives explanations to artists on how to mix paint correctly, which well reflect the state of the art in colorimetry at the time : there exist only two primary color components, white and black, from which we can obtain noble secondary colors, orange, green and purple. All other colors are obtained by mixing these three components.

2.1.3.2 Newton and the prism

In 1672, Sir Isaac Newton (1642 - 1726) publishes the result of experiments conducted since the late 1660s on the prism, in which he tries to understand how colors actually appears in the rainbow. Newton will discover diffraction and will bring a clear proof that, contrarily to the understanding physicists had of colors at his time, all colors are already contained in white light, and that this is not the prism which colors light. In his demonstration, he basically separates colors from white light using a prism, and, most important, shows that when mixing back all colors of the rainbow, we obtain white light.

2.1.4 Physical nature of light : corpuscular or wavelike?

Once the question about the speed of light definitively answered, the debate shifted on the physical nature of light. Different theories were proposed through history.

2.1.4.1 Newton and the corpuscular theory

In 1675, after his experiments on diffraction, Newton publishes his *Hypothesis of Light*, in which he states that light is composed of a high number of particles of high elasticity. In the meantime, Robert Hooke (1635 - 1703), one of the greatest british experimental scientists and rival of Newton, maintained that light behaved much more like a wave. Newton's main arguments were that, on the one hand, light could be reflected, similarly to particles bouncing on a surface, and on the other hand, light could not bend around objects like waves usually do. We must note that at Newton's time, refraction of light passing from a medium to another of different density had not yet been explained.

2.1.4.2 The wave theory

In 1678, Christian Huygens (1629 - 1695), a famous dutch scientist, publishes his *Traité de la lumière*, in which he writes what is known as the Huygens Principe. According to Huygens, each point of a wavefront behaves like a source of small secondary wavelets. At the next instant, the front boundaries enveloppe of the wavelets forms the new wavefront. The Huygens principe allows to predict the propagation direction of a wavefront.

Thomas Young (1773 - 1829) proves once and for all, in 1803, that light does definitely not behave like particles, because it posesses a property only specific to waves : being subject to interferences. The idea is simple : when two light beams encounter, the two wavefronts add-up, where particles would collide.

2.1.4.3 Maxwell and the electromagnetic theory

James C. Maxwell (1831 - 1879) is considered by many as the scientist from the nineteenth century who has kept the biggest influence at the twentieth century. In 1865, Maxwell published a synthesis of previous works by Gauss, Ampere and Faraday on electricity and magnetism, which were slightly modified by the addition of a new term in their equations, to solve the problem of the conservation of charge. This work led to the prediction of the existence of electromagnetic waves, which had not been discovered yet. Most important, Maxwell's equations enabled to calculate the speed of electromagnatic waves, and, by discovering that these waves traveled at the speed of light, he had just found a set of equations that unify electricity, magnetism and optics as all dealing with problems related to electromagnetism. For the first time, light was mathematically described as made of electromagnetic waves.

2.1.4.4 Planck and the quantum theory

Maxwell's theory of light as waves appeared to do not explain completely some aspects of light, such as the relation between the properties of incoming light rays and the energy it will brough on a surface being illuminated. According to Maxwell, that imparted energy depends only on their intensity, although various experiments did show it would rather be a function of the incoming light's frequency.

2.2 Common light models

In today's physics, light is still defined as a phenomenon exhibiting both properties of particles and waves. Since a fully comprehensive optical model would be too complex to apply in practice, optical scientists instead actually use simplified models that can account for most phenomena related to light.

2.2.1 Geometrical optics

The geometrical model of light is the most commonly used in practice in several applications related to the study of behaviour of light and its interactions with objects.

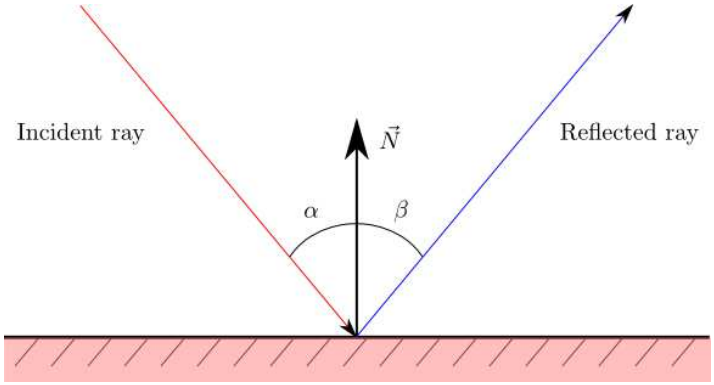

Figure 2.3: An incident ray of light is reflected on a surface. If the surface behaves like a perfect mirror, the angle of incidence α and reflection β should be identical.

In geometrical optics, questions related to the very nature of light are not considered, and in such cases physical models seem more appropriate. Light is strongly approximated as composed of rays who's behaviour is studied in different environments. This approximation still makes it possible to study the main phenomenons such as reflection and refraction.

When travelling in an homogeneous and constant environment, light moves in straight line. As the material's optical density changes, so does the speed of light in the medium.

When light hits an object or a participating medium particle, two behaviors are possible for each single light ray : reflection and refraction. Light emitted from a source in a particular direction is composed of billions of light rays, some will be reflected and some refracted. Another phenomenon which has to be considered is the absorption of part of the ray's energy at each interaction with a surface.

2.2.1.1 Reflection

The laws of reflection (figure 2.3) of the light on a planar surface state that :

- The incident ray, the normal vector to the reflecting surface, and the outgoing reflected light ray are coplanar.
- The angle of reflection between the normal and the reflected ray is equal to the angle of incidence between the incoming ray and the normal.

A material which equally scatters light in all directions whatever the incident angle is called *perfectly diffuse*. A material which ideally reflects light according to the laws of reflection described above is called *perfectly specular*.

In everyday's life, although these laws are absolutely valid, perfect reflections can barely be observed, except on mirrors, which are almost perfectly specular surfaces. When observed at small scales, most surfaces are not perfectly planes, and will always slightly diffuse light around the main specular reflection direction.

Figure 2.4: Due to refraction, light rays bend when exiting the water back into the air, which makes this straw appear broken.

2.2.1.2 Refraction

Refraction is the phenomenon occurring when a light ray travelling inside a material M_1 encounters a second material M_2 and, instead of being reflected, penetrates and continues its propagation inside M_2.

Although light keeps travelling in straight line while staying inside the same material, when passing from one material to the other (figure 2.5), light rays are bent according to Snell's law of refraction. This deviation is strong enough to be noticeable visually in everyday's life (figure 2.4). Unlike with reflection, light is refracted in a different way depending on the nature of the refracting material. Each different material has a specific refractive index n (see figure 2.1), which is defined as the ratio between the speed of light in vacuum c and the speed of light v in the material.

Note : In more comprehensive optical models, we note that the speed of light in a medium depends on the wavelength on the light, and from equation 2.1, we see that so does the refractive index of the material. In geometrical optics, by *refractive index*, we refer to the absolute refractive index of the medium, noted n_D, commonly defined as the refractive index for the helium yellow line which wavelength is $\lambda = 587.6$nm, close to the middle of the visible spectrum.

$$n = \frac{c}{v} \tag{2.1}$$

Snell's law of refraction enounces the existing relation between refraction indices n_1 and n_2 of materials M_1 and M_2, with the angle of incidence θ_1 and the angle of refraction θ_2, and is defined as follows :

$$n_1 * \sin(\theta_1) = n_2 * \sin(\theta_2) \tag{2.2}$$

We can see from Snell's relation that when $\theta_1 = 0$ and the incident ray is perpendicular to the refraction surface (i.e. the boundary between the two media), outgoing rays do not bend and stay colinear to the normal to the surface. On the contrary, as the light is inclined when it hits the surface and θ_1 increases, rays are deviated even more that the difference between n_1 and n_2 is strong.

The reason why light bends when it enters a medium of different optical density is linked to equation 2.1, which states that the speed of light v in a medium is linearly proportional to its

Material type	Refractive index
Vacuum	$n = 1.0$
Air	$n = 1.0003$
Ethyl alcohol	$n = 1.36$
Water	$n = 1.33$
Crown glass	$n = 1.52$
Flint glass	$n = 1.66$
Diamond	$n = 2.42$

Table 2.1: A few materials and their refractive index

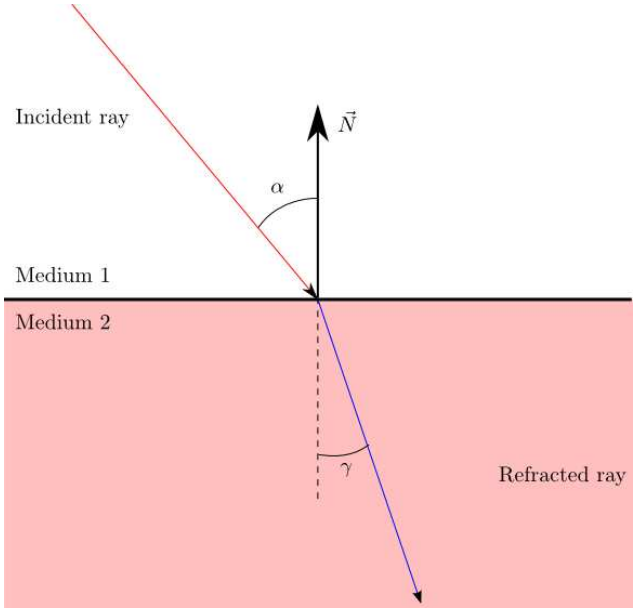

Figure 2.5: When a light ray is transmitted from medium 1 to medium 2, it bends according to Snell's law (2.2). When the two media are different, the angle of incidence α is not equal to the angle of refraction γ.

refractive index n, itself linked to its optical density. Indeed, when light enters a medium of higher refractive index, rays decelerate. We consider the example of light arriving on the surface of a translucent object with a direction that is not colinear with the normal to the surface. Exactly like a car going fast would instantly turn to the right when the driver only activate its right wheels brakes, light is deviated towards the direction perpendicular to the surface because some light particles will be slown down before their neighbours themselves touch the refracting material.

Refraction can sometimes result in a reflection of light on the boundary between two refractive media M_1 and M_2. This situation occurs when the incident angle is such that the refracted angle of rays supposed to penetrate into M_2 happen to be greater than $\frac{\pi}{2}$. When the refractive index of M_1 is actually greater than that of M_2, this phenomenon is called *internal reflection*.

2.2.2 Physical optics

Contrarily to geometrical optics which was a too simplified model, physical optics consider light as a wave, and thus can account for effects such as diffraction and interference.

2.2.2.1 The Huygens-Fresnel principle

Christian Huygens (1629 - 1695), a dutch mathematician, physicist and astronomer, wrote in his famous *Traité de la lumière*, what is known as the Huygens principle. He proposed that every point illuminated by a light wavefront at time t itself becomes a source of spherical secondary light wavelets. The overall aspect of the wavefront a time $t + 1$ is the resulting surface tangent to the outer side of all these wavelets.

Although this principle can explain such phenomenons as reflection and refraction, it does not answer the question why, if the secondary wavelets themselves propagate in all directions, no backward wavefront appears. Why does the wavefront continue to only propagate away from the source ?

Later, Augustin Fresnel (1788 - 1827) added to Huygens' postulate that, assuming that all wavelets have the same frequency, the amplitude at a given point on the wavefront at time $t + 1$ is the sum of all wavelets superimposing on that point. He also brought mathematical support to Thomas Young's demonstration that unlike particles, light as a wave was subject to interferences. Fresnel approached a first answer to the problem of the hypothetical backward propagation by recognizing the need for an angular dependence of the wavelets. Later, Kirchhoff (1824 - 1887) provided a mathematical expression of the phenomenon of diffraction occurring when a wavefront passes through a small opening in an occluding surface. He also defined an obliquity factor which better solved the question of the backward propagation in Fresnel's principle.

2.2.2.2 Diffraction

The phenomenon of diffraction is specific to waves, and therefore cannot be explained with geomerical optics, where light is modeled as rays which propagate in straight line in homogeneous media.

We illustrate this phenomenon using sound waves, which behave like light waves when it comes to illustrate diffraction. We consider an environment composed of two large separate rooms linked with a small open doorway, where one person is standing in each room. When the first person

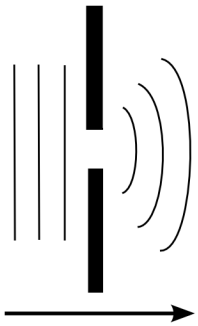

Figure 2.6: Diffraction of a wavefront by a small opening. Due to the Huygens-Fresnel principle, instead of continuing their way parallely to the obstacle, the waves behave as if they were emitted by a source situated near the hole.

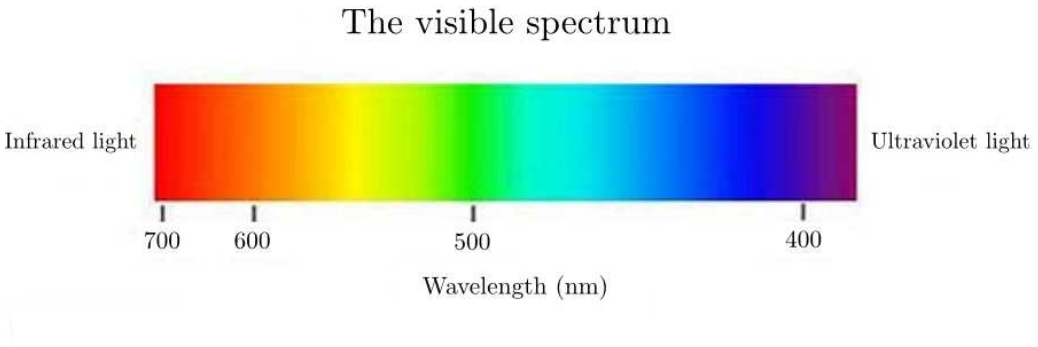

Figure 2.7: Color wavelengths.

speaks in one room, because of diffraction, the second person will ear the sound in the other room as if it was emitted from the doorway, regardless of their respective position in the rooms.

Similarly, we consider a light wavefront advancing perpendicularly towards a wall and eventually passing through a small opening in its center (figure 2.6). It will not result in straight light rays continuing their way in the sole lighting direction, but rather in hemispherical light waves having their highest amplitude in front of the hole in the propagation direction.

Diffraction is explained by the Huygens-Fresnel principle, which states that the small portion of waves passing through the opening will act themselves as sources for secondary wavelets propagating in all directions on the other side of the diffracting obstacle.

2.2.2.3 Wavelength and color

The visual appearance of light, its color, is intrinsically linked to its wavelength, i.e. the minimum distance between two different peaks on a periodic light wave.

Only a small portion of the electromagnetic spectrum is visible to humans (table 2.2 and figure 2.7), namely light which wavelength is comprised between 350 and 700 nanometers.

Wavelength	Wave type
$\lambda < 0.01\text{nm}$	Gamma rays
$\lambda \in [0.01\text{nm}, 10\text{nm}]$	X-rays
$\lambda \in [10\text{nm}, 350\text{nm}]$	Ultraviolet waves
$\lambda \in [350\text{nm}, 700\text{nm}]$	Visible light
$\lambda \in [700\text{nm}, 1\text{mm}]$	Infrared waves
$\lambda \in [1\text{mm}, 30\text{cm}]$	Microwaves
$\lambda > 30\text{cm}$	Radio waves

Table 2.2: Light and the electromagnetic spectrum

Wavelength	Color
$\lambda \in [350\text{nm}, 450\text{nm}]$	Violet
$\lambda \in [450\text{nm}, 475\text{nm}]$	Blue
$\lambda \in [475\text{nm}, 495\text{nm}]$	Cyan
$\lambda \in [495\text{nm}, 570\text{nm}]$	Green
$\lambda \in [570\text{nm}, 590\text{nm}]$	Yellow
$\lambda \in [590\text{nm}, 620\text{nm}]$	Orange
$\lambda \in [620\text{nm}, 700\text{nm}]$	Red

Table 2.3: Main colors in the visible spectrum

2.2.2.4 Dispersion

Dispersion of light is the phenomenon occurring when a polychromatic light is refracted, revealing its different monochromatic components. The famous typical examples are the dispersion of white light through a prism (figure 2.8), or the dispersion of sunlight through rain leading the the formation of a rainbow (figure 2.9).

The phenomenon of dispersion is actually caused by the refraction of all monochromatic components, each leaving the refracting medium in a different direction. Each component having a different wavelength and therefore a different refractive index, the refraction makes it bend with a different angle as well. The shorter the wavelength, the higher the refractive index is.

2.2.3 Quantum optics

Quantum optics is today's most comprehensive optical model, light is studied as possessing a true wave-particle duality.

2.2.3.1 Wave-particle duality

When Huygens was trying to impose his theory of light as a wave against Newton's particle model, electromagnetic waves had not been discovered yet. Although electromagnetic waves can propagate in vacuum, classical mecanic waves, the only wave type known by Huygens and his contemporary physicists, require a medium to propagate, whether solid, liquid or gas. To come

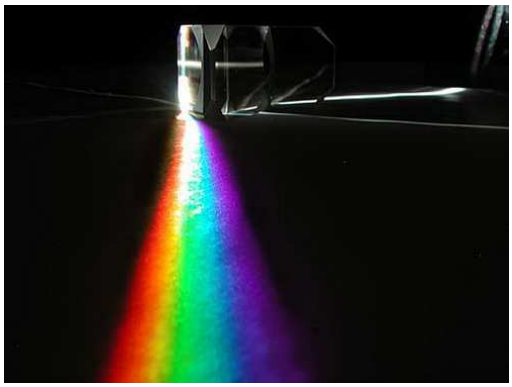

Figure 2.8: The white light contains all colors of the visible spectrum, which can be dispersed through a prism.

Figure 2.9: An example of diffraction that can be found in the nature : the rainbow.

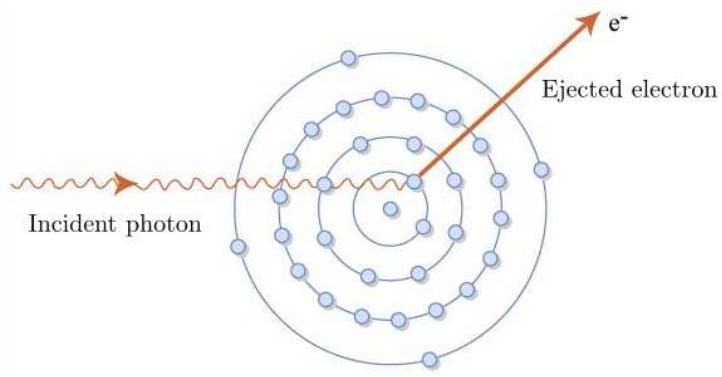

Figure 2.10: The photoelectric effect : when a photon strikes an atom, some of its orbiting electrons may be ejected.

with an explanation about how light could travel through outer space, Huygens postulated that the Solar System was evolving through a luminiferous aether, an hypothetical unknown invisible matter that could propagate light. The view of light as a wave was reinforced in 1801 by Thomas Young's double-slit experiment which made it clear that light shared some properties with waves.

Between 1881 and 1887, the famous Michelson-Morley series of experiments aimed at detecting the effects on the speed of light of the presence of the aether around the moving earth. Light rays were emitted from the ground both perpendicularly to and in the direction of the earth's rotation. They were then reflected on mirrors and re-directed toward an observer on the ground. Because the aether was supposed to be static relatively to the earth which was rotating, if light basically behaved like mechanical waves emitted from a moving source inside a static medium, according to the velocity addition law, it should have taken longer to light to propagate colinearly than perpendicularly to the rotation direction. The experiments all resulted in negative results, indeed it took the same time for light to travel along the same distance no matter the orientation of the rays. From this time, physicists began to doubt the existence of the luminiferous aether and, at the beginning of the twentieth century, the question of the nature of light remained unanswered.

2.2.3.2 The photoelectric effect

A particle-only theory of light explained how light could travel through vacuum, but was still in contradiction with compelling observations that light was also subject to wavelike phenomena, such as interferences and diffraction. This wave-particle duality behaviour seemed not random, but rather occurred steadily under any conditions and in a perfectly measurable and reproductible fashion.

The year 1905 is called *annus mirabilis* (miracle year), due to the high number of discoveries made by Albert Einstein (1879 - 1955) in a few months time. Einstein first invented the quantum theory of light, in which light is considered as made of small particles of energy, later called photons, in 1924.

The photoelectric effect (figure 2.10), first observed in the early 1800's, challenged the current knowledge of the nature and behavior of light. When a light source is directed towards a metallic surface, a varying quantity of electrons is released from the surface. The aspects of this phenomenon

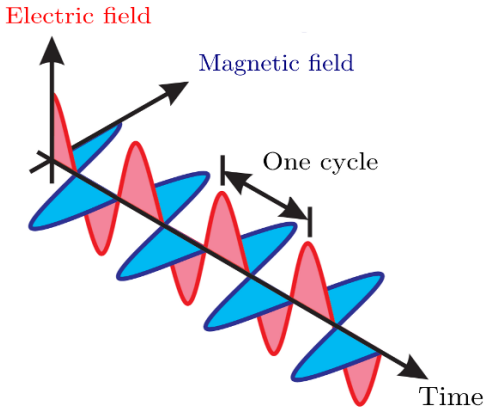

Figure 2.11: Photons are electromagnetic particles, oscillating in two perpendicular directions due to both an electric and a magnetic field.

which defied understanding were the fact that the energy beared by the released electrons did not depend on the intensity of the incident light, and also the fact that regardless of the intensity of the source, the incident light had to exceed a certain frequency to trigger the photoelectric effect.

Einstein published four ground-breaking articles, starting with an explanation of the photoelectric effect, in which he postulated that in a light wavefront, the energy is not equally beared by all points on the wavefront, but rather by small particles of light energy. He then explains that the release of an electron only occurs on specific situations where it actually collides with a photon arriving on the metallic surface.

2.2.3.3 Properties of a photon

A photon is an elementary corpuscular particle of light, with a mass and a charge both equal to zero. In 1900, Max Planck postulates that exchanges of energy between a light radiation and matter can only occur by small packets, called quanta, containing as much energy that the radiation's frequency is high. The quantum is the minimum finite quantity of energy that can be exchanged.

In 1926, Louis de Broglie (1892 - 1987), with his law about the particle-wave duality, generalized the works by Planck and Einstein. This way, quantum physics reconciliated the corpuscular aspect of light, with photons as light's particles containing energy, and its wave aspect, since light is considered as an electromagnetic radiation (figure 2.11).

We finish with the main properties of photons :

- In vacuum, photons propagate at $c \approx 2.99792458 \times 10^8 \text{m.s}^{-1}$, the speed of light.
- The energy E beared by a photon of frequency ν is obtained by :

$$E = h * \nu, \tag{2.3}$$

where h is Planck's constant, with value $h \approx 6.62606957 \times 10^{-34}$J.s, according to the CODATA of 2006.

- From this expression we can notice that the energy of a photon is proportional to its frequency.
- Photons have no rest mass : $m_0 = 0$.
- The dynamical mass of a photon is obtained by :

$$m = \frac{E}{c^2} = \frac{h * \nu}{c^2} \tag{2.4}$$

- The momentum of a photon equals :

$$p = \frac{h * \nu}{c} \tag{2.5}$$

Chapter 3

The equation of transfer

"Science, my lad, is made up of mistakes, but they are mistakes which it is useful to make, because they lead little by little to the truth."

Jules Verne, *Journey to the Center of the Earth*

3.1 Balancing the energies

In physical optics, light is considered as composed of microscopic particles, called photons, which bear a small quantity of energy varying with the color, i.e. the frequency of the light.

The intensity of the radiance lighting a surface is proportional to the concentration of photons hitting every small elementary area of the surface. A surface is a two-dimensional entity. In comparison, a participating medium is, by definition, a volumetric object, but the idea remains the same : the intensity of light travelling through a medium depends directly on the number of photons inside a given volume.

The transport equation describes the interactions between light and a scattering medium. More precisely, it formulates how the quantity of photons entering and exiting a differential portion of the scattering medium are balanced.

3.1.1 The phase space

Definition

Like Arvo in [Arv93], we introduce the *phase space*, in which we will represent photons by their position together with their other properties that change from one particle to another. If we assume that their speed remain constant, our photons actually possess two properties that change over the time, corresponding to five degrees of liberty.

Since they are light particles evolving in a three-dimensional space, they first have at least one degree of liberty for each coordinate x, y and z. Moreover, as the manner in which they interact

with objects also depends on their incident direction, if we express this travel direction in spherical coordinates, we add two more degrees of liberty, one for each angle θ and ϕ.

Such a representation is interesting in our equations because all photons around the same position in $\mathbb{R}^3$ and directed towards the observer will lie in the same neighborhood in the phase space.

We introduce $\Psi(r, \omega)$, the unitary domain of the phase space constituted of the one cubic meter neighborhood centered in $r \in \mathbb{R}^3$, and the one steradian solid angle centered in direction $\omega \in \mathrm{S}^2$.

The phase space volume

By similarity to the notion of volume (in m^3) in three dimensions, which measures the quantity of space contained in a delimited portion of the scene, we can define the phase space volume as the measure of the size of a portion of the phase space, delimited by boundaries along all five dimensions. It is expressed in $m^3.sr$ as the product of a spatial volume (in m^3) and an area on the surface of an unitary sphere (in steradians, written sr).

By definition, $\Psi(r, \omega)$ has a fixed phase space volume of one $m^3.sr$.

The angular density of photons

The angular density corresponds to the number of photons in $\Psi(r, \omega)$. It is written $n(r, \omega)$, where r is a position in $\mathbb{R}^3$ and ω is a direction on the sphere in S^2.

Since it is expressed in photons per $m^3.sr$, this quantity is linearly proportional to the volume of the domain around r and the measure of the solid angle around ω.

The phase space flux

Based on n is the phase space flux ϕ, i.e. the quantity of photons crossing an unitary phase space volume per unit of time :

$$\phi(r, \omega) = v.n(r, \omega), \tag{3.1}$$

where v is the speed of photons in $m.s^{-1}$, and $n(r, \omega)$ is the number of photons in the phase volume in $m^{-3}.sr^{-1}$. The flux itself is expressed in $m^{-2}.sr^{-1}.s^{-1}$.

Since the flux of photons is closely linked with the properties and the geometry of the medium, its value may vary with the position r and the orientation ω. In a simulation, the flux would be barely computable analytically, and would rather be estimated based on the distribution of the light at the previous states.

3.1.2 Emission and absorption

The emission function

If the participating medium itself behaves as a light source, it generates new photons that add up to its energy. The intensity of the light source is related to the rate, i.e. the rapidity with which it emits new photons.

If the intensity of the light remains constant over the time, the emission function varies inside the phase space. It is written $q(r, \omega)$ and represents the number of photons emitted per unitary phase volume $\Psi(r, \omega)$ per second, i.e. in $m^{-3}.sr^{-1}.s^{-1}$.

The absorption coefficient

When a photon travels through a scattering medium, it may simply pass through unblocked or collide with particles. In this case, it may bounce in another direction or be absorbed, and have its energy transformed into heat.

The absorption coefficient is written $\sigma_a(r)$ and expresses the probability for a photon to be absorbed when travelling through an unitary distance inside the medium. It does not depend on the incoming direction ω, since medium particles may block photons regardless of where they arrive from. The value is given as the percentage of photons absorbed within r per meter.

The quantity of photons absorbed in each unitary phase space domain $\Psi(r, \omega)$ per second is given by :

$$\mathrm{C}_{\mathrm{abs}}(r, \omega) = \sigma_a(r)\phi(r, \omega) \tag{3.2}$$

3.1.3 The scattering function

The scattering function

The portion of light which interacts with the medium but fails to be absorbed is to be scattered in different directions.

The scattering function is written $k(r, \omega_{\mathrm{in}}, \omega_{\mathrm{out}})$ and represents the probability for a photon arriving at position r from direction ω_{in} to be scattered towards a direction comprised within the one steradian solid angle centered in ω_{out}, when travelling through the medium along a distance of one meter. This probability is to be understood as a percentage of photons scattered at r, from ω_{in} towards ω_{out} per meter per steradian, in $m^{-1}.sr^{-1}$.

In computer illumination in general, and in this thesis, we also refer to the *phase function*, written $F(r, \omega_{\mathrm{in}}, \omega_{\mathrm{out}})$. The phase function takes the same three arguments and plays a role similar to $k(r, \omega_{\mathrm{in}}, \omega_{\mathrm{out}})$, the difference being that $F(r, \omega_{\mathrm{in}}, \omega_{\mathrm{out}})$ does not take into account the probability of behaviors other than reflection. When the scattering function $k(r, \omega_{\mathrm{in}}, \omega_{\mathrm{out}})$ indicates the probability of light bouncing in a particular direction among all possible events (absorption, etc.), $F(r, \omega_{\mathrm{in}}, \omega_{\mathrm{out}})$ gives a percentage among all rays whom we already know will reflect.

The scattering function already takes into account the phase function, and therefore only depends on the phase angle between the incoming direction and the outgoing direction.

In-scattering

When computing the appearance of a participating medium as viewed from the camera's point of view, only photons situated on the view ray and directed toward the camera will contribute to the final color of each pixel.

In-scattering is the phenomenon which makes it possible to visualize a participating medium illuminated by a light source. A photon travelling within the scene or directly coming from the

source finds itself scattered by the medium at a position situated on the view ray, and happens to be re-directed towards the camera.

We consider the unitary phase space domain $\Psi(r, \omega)$, and we note Ω the unitary solid angle. Just before colliding with the medium, this photon was initially lying outside $\Psi(r, \omega)$ because arriving from direction $\omega_{\text{in}} \notin \Omega$. After the scattering collision, it is deflected at position $r \in V$ in a new direction $\omega \in \Omega$, thus being inserted into $\Psi(r, \omega)$. By this process, new energy is added to the radiance reflected by the nearest solid object along the way towards the observer.

We now want to sum up all photons arriving around r from all directions ω_{in} not included in Ω, and which may be scattered inside $\Psi(r, \omega)$, indeed towards the unitary solid angle around ω.

We have :

$$\mathrm{C}_{\text{in}}(r, \omega) = \int_{\mathrm{S}^2 - \Omega} k(r, \omega_{\text{in}}, \omega) \phi(r, \omega_{\text{in}}) d\omega_{\text{in}}, \tag{3.3}$$

with $\mathrm{C}_{\text{in}}(r, \omega)$ in $m^{-3}.sr^{-1}.s^{-1}$.

Out-scattering

Out-scattering, by opposition, sees photons moving in direction of the camera being deflected in another direction after an interaction with the medium. Even if the out-scattered photon stays inside the medium itself, this phenomenon removes radiance from the light which will ultimately reach the camera.

We want to integrate the number C_{out} of photons travelling inside $\Psi(r, \omega)$ and deflected out of it per second.

We have :

$$\mathrm{C}_{\text{out}}(r, \omega) = \int_{\mathrm{S}^2 - \Omega} k(r, \omega, \omega_{\text{out}}) \phi(r, \omega) d\omega_{\text{out}}, \tag{3.4}$$

where $\mathrm{S}^2 - \Omega$ is the solid angle domain on the sphere towards which deflected photons leave $\Psi(r, \omega)$.

3.1.4 The transport equation

3.1.4.1 Building the equation

Domain of study

To build the transport equation, we first need to choose a domain of study in terms of time and space. The most convenient domain is obviously the unitary phase space domain, the equation can then be integrated to summarize all the phenomenon happening over the whole medium, or derivated to balance the energies more locally.

We consider the flow of photons in one cubic meter of space and one steradian of orientation, and one second of time. By definition, because they are in constant motion, all photons entering our domain of study have to leave it due to one phenomenon or another. The idea behind the transport equation is to separate all phenomena between those who remove photons from the flow on the left side and those who bring new photons into the flow on the right side.

Accounting for streaming

Last but not least, we have to account for photons which enter and exit the unitary phase volume $\Psi(r, \omega)$ simply by crossing the bounding surface of V, and whos orientation is included in Ω. If we do not account for photons streaming in and out the volume, it is impossible to reach the perfect equilibrium between the left term and the right term of the equation. The reason for this is that interactions between photons and the medium are not supposed to be equilibrated in the general case. The medium always, at least, absorbs and scatters small quantities of light. If we do not have two quantities that we know are balanced, we are unable to express any two term equation of any form.

We can divide streaming in two quantities : in-streaming, for photons crossing the border to enter V, and out-streaming, for photons exiting V. We note that these two quantities naturally account for photons that will just enter or exit the volume without colliding with the medium, indeed without being neither in-scattered or out-scattered, nor absorbed.

Streaming of light inside or outside a volume is a phenomenon which is very distinct from absorption, emission and scattering since it only occurs on the bordering surface of the volume and not in the same manner at each point throughout the volume. Therefore, we begin by expressing the total number of photons $\mathrm{St}_{\mathrm{in,\ V}}$ entering and $\mathrm{St}_{\mathrm{out,\ V}}$ exiting our unitary phase volume $\Psi(r, \omega)$ per second.

We have :

$$\mathrm{St}_{\mathrm{in,\ V}}(s, \omega) = \int_{\Omega} \int_{\partial V} \phi(r, \omega) \min[\omega \cdot \mathrm{n}(s), 0] ds d\omega \tag{3.5}$$

and

$$\mathrm{St}_{\mathrm{out,\ V}}(s, \omega) = \int_{\Omega} \int_{\partial V} \phi(r, \omega) \max[\omega \cdot \mathrm{n}(s), 0] ds d\omega, \tag{3.6}$$

where $\mathrm{n}(s)$ is the normal vector at position s on the bordering surface ∂V of volume V.

The flux $\phi(r, \omega)$ corresponds to the stream of photons through a surface perfectly orthogonal to ω. Based on the same principle as Lambert's cosine law, the dot product $\omega.\mathrm{n}(s)$ modulates the flux to fit the local orientation of ∂V at position s.

Then, we know that because light cannot flow in both directions at the same position s on ∂V, the distribution of zeros and non-zero values in $\mathrm{St}_{\mathrm{in,\ V}}$ and $\mathrm{St}_{\mathrm{out,\ V}}$ is perfectly complementary. Indeed, we can sum in-streaming and out-streaming together in a single global streaming quantity St_V, representing the global balance between light flowing in and light flowing out of $\Psi(r, \omega)$ naturally.

We obtain :

$$\mathrm{St}_V(s, \omega) = \int_{\Omega} \int_{\partial V} \phi(r, \omega) \omega \cdot \mathrm{n}(s) dr d\omega \tag{3.7}$$

We now want to transform the integration over the surface ∂V to a classic integration over V, we would then be able to remove both integrals and express streaming in its differentiated form. This is made possible using the *divergence theorem*.

We now have :

$$\mathrm{St}_V(s,\omega) = \int_\Omega \int_V \omega \cdot \nabla\phi(r,\omega) dr d\omega \tag{3.8}$$

Finally, we remove the integrals and come with the differentiated form :

$$\mathrm{St}(s,\omega) = \nabla \cdot [\omega\phi(r,\omega)] = \omega \cdot \nabla\phi(r,\omega), \tag{3.9}$$

with $\mathrm{St}(s,\omega)$ in $m^{-3}.sr^{-}1.s^{-1}$.

Basic formulation of the equation

We can now express the transport equation under its basic form. Explicitely, it states that all energy outgoing (or transformed, in the case of the absorption) from an unit portion of the phase space (streaming, absorption and out-scattering) is perfectly equivalent to the energy entering or emitted in this portion (emission and in-scattering). We have :

$$\mathrm{St}(s,\omega) + \mathrm{C}_{\mathrm{abs}}(r,\omega) + \mathrm{C}_{\mathrm{out}}(r,\omega) = q(r,\omega) + \mathrm{C}_{\mathrm{in}}(r,\omega) \tag{3.10}$$

By replacing each quantity by its own expression, we finally obtain :

$$\begin{aligned} \omega \cdot \nabla\phi(r,\omega) + \sigma_a(r)\phi(r,\omega) + \int_{\mathrm{S}^2-\Omega} k(r,\omega,\omega_{\mathrm{out}})\phi(r,\omega) d\omega_{\mathrm{out}} \\ = q(r,\omega) + \int_{\mathrm{S}^2-\Omega} k(r,\omega_{\mathrm{in}},\omega)\phi(r,\omega_{\mathrm{in}}) d\omega_{\mathrm{in}} \end{aligned} \tag{3.11}$$

3.1.4.2 Simplifying the equation

Simplification : integrating $\mathrm{C_{in}}$ and $\mathrm{C_{out}}$ over S^2

The quantities of in-scattered and out-scattered photons $\mathrm{C}_{\mathrm{in}}(r,\omega)$ and $\mathrm{C}_{\mathrm{out}}(r,\omega)$ are both defined based on an integration over the domain $\mathrm{S}^2 - \Omega$. For C_{in}, we need to omit photons which come with a direction already included inside Ω, because even if they are scattered inside Ω, those photons were already counted as part of $\Psi(r,\omega)$. For $\mathrm{C}_{\mathrm{out}}$, by definition, we need to omit photons which are deflected back inside Ω.

However, we can notice that we actually omit exactly the same term in both C_{in} and $\mathrm{C}_{\mathrm{out}}$:

$$\int_\Omega k(r,\omega_{\mathrm{in}},\omega)\phi(r,\omega_{\mathrm{in}}) d\omega_{\mathrm{in}} = \int_\Omega k(r,\omega,\omega_{\mathrm{out}})\phi(r,\omega) d\omega_{\mathrm{out}} \tag{3.12}$$

Indeed, light "in-scattered" from inside Ω corresponds exactly to light "out-scattered" back inside Ω. It is therefore possible, in order to simplify the notation, to extend the integration domain to S^2 in both quantities without changing the meaning of the whole transport equation.

We have :

$$\begin{aligned} \omega \cdot \nabla\phi(r,\omega) + \sigma_a(r)\phi(r,\omega) + \int_{\mathrm{S}^2} k(r,\omega,\omega_{\mathrm{out}})\phi(r,\omega) d\omega_{\mathrm{out}} \\ = q(r,\omega) + \int_{\mathrm{S}^2} k(r,\omega_{\mathrm{in}},\omega)\phi(r,\omega_{\mathrm{in}}) d\omega_{\mathrm{in}} \end{aligned} \tag{3.13}$$

Simplification : merging σ_a and C_{out}

First of all, we introduce the scattering coefficient $\sigma_s(r)$, as the local probability for a photon arriving at r to be scattered, when travelling along a distance of one meter.

$$\sigma_s(r) = \int_{S^2} k(r, \omega_0, \omega_{\text{out}}) d\omega_{\text{out}} \tag{3.14}$$

Then, we can write :

$$C_{\text{out}}(r, \omega) = \phi(r, \omega)\sigma_s(r) \tag{3.15}$$

We extracted $\phi(r, \omega)$ from inside the integral of C_{out} and simply placed it in front as a factor of this integral. Because we integrate all over S^2, the parameter ω, whatever its value, does not change anything in the equation anymore. It can therefore be removed from the parameters needed by the integral, and be replaced in the equation by an arbitrarily fixed direction, which, whatever its value, will fit in the equation.

We now define $\sigma(r)$ as the global interaction coefficient, indeed the probability for a photon to interact at all with the medium when travelling along one meter :

$$\sigma(r) = \sigma_a(r) + \sigma_s(r) \tag{3.16}$$

Standard formulation of the transport equation

We finally obtain the standard simplified version of the transport equation :

$$\omega \cdot \nabla\phi(r, \omega) + \sigma(r)\phi(r, \omega) = q(r, \omega) + \int_{S^2} k(r, \omega_{\text{in}}, \omega)\phi(r, \omega_{\text{in}}) d\omega_{\text{in}}$$

3.2 The equation of transfer

3.2.1 Introducing radiance

Radiance as a flux of energy

In quantum optics, light is composed of electromagnetic particles, called photons. Each photon transports a small quantity of energy, i.e. a small electromagnetic radiation, in Joules (J).

$$E = hv, \tag{3.17}$$

where h is the Planck constant, in $J.s$, which establishes the relationship proportionality between the energy E of a photon, in J, and its frequency v of electromagnetic vibration, in Hertz (Hz).

We already defined the flux ϕ as the number of photons crossing an unitary phase volume by streaming per second. In terms of surface, this corresponds to the quantity of photons crossing a surface perpendicular to the direction of the flux, per square meter of surface area, per steradian of solid angle, per second.

If a quantity of energy is associated to each photon, we can express the radiance $L(r, \omega)$ as a flux of energy in the phase space :

$$L(r,\omega) = E\ \phi(r,\omega) \tag{3.18}$$

$$L(r,\omega) = hv\ \phi(r,\omega) \tag{3.19}$$

The value $L(r,\omega)$ is the radiometric energy received in Joule per phase surface area per second, i.e. $\frac{J}{m^2.sr.s}$.

We define an equivalent to the emission function in terms of radiance, associated to volumetric sources. The volume radiance emission function $\varepsilon(r,\omega)$, being the radiance emitted per unitary phase volume $\Psi(r,\omega)$ per second, is written as :

$$\varepsilon(r,\omega) = hv\ q(r,\omega), \tag{3.20}$$

in $\frac{J}{m^3.sr.s}$.

Radiance in the transport equation

When we insert the radiance quantities in the transport equation, we obtain :

$$\omega \cdot \nabla L(r,\omega) + \sigma(r)L(r,\omega) = \varepsilon(r,\omega) + \int_{\mathrm{S}^2} k(r,\omega_{\mathrm{in}},\omega)\varepsilon(r,\omega_{\mathrm{in}})d\omega_{\mathrm{in}} \tag{3.21}$$

3.2.2 Radiance emitted by the surface

The purpose of the equation of transfer is to express the radiance $L(r,\omega)$ perceived by an observer or a camera situated at position r and looking in direction $-\omega$. Contrarily to the transport equation 3.17, the equation of transfer is not expressed in the form of an equilibrium between energies incoming and outgoing of a volume, but is rather similar to a function providing a result directly exploitable.

We have to account for two distinct entities which bring their contribution to the radiance received in r :

- The nearest solid surface visible from r in the direction $-\omega$.
- The scattering medium, more precisely the portion of the medium traversed by the view ray and situated between r and the surface.

Surfacic equivalents of volumetric quantities

In theory, a surface is not a volumetric object like a participating medium, but can be assimilated to a bounded plane in the three-dimensional space. Therefore, our transport equation, defined for three-dimensional neighborhoods inside the medium, cannot be applied directly to surfaces.

We first introduce a surface radiance emission function, written $\varepsilon_b(s,\omega)$, and defined by :

$$\varepsilon_b(s,\omega) = hv\ q_b(s,\omega), \tag{3.22}$$

in $\frac{J}{m^2.sr.s}$.

$\varepsilon_b(s,\omega)$ is the radiance emitted per square meter of surface area around s, per steradian of solid angle, and $q_b(s,\omega)$ is the equivalent of $q(s,\omega)$ for surfaces.

We also define $\gamma(r,\omega) \in \mathbb{R}^3$, as the position of the projection of r on the nearest surface in direction ω, so that radiance outgoing from a surface in the direction ω and arriving at s originates from position $\gamma(r,-\omega)$.

Finally, we define $k_b(s,\omega_{\text{in}},\omega_{\text{out}})$, the surfacic scattering function, representing the percentage of all radiance incoming on the surface at s from ω_{in} which will be scattered towards ω_{out}, per steradian of angular neighborhood around ω_{out}.

Radiance $\Gamma(s,\omega)$ outgoing from the surface

We can now express the first term of the equation of transfer : the radiance $\Gamma(s,\omega)$ outgoing from a surface at position s towards direction ω.

$$\Gamma(s,\omega) = \varepsilon_b(s,\omega) + \int_{H_S^-} k_b(s,\omega_{\text{in}},\omega) L(s,\omega_{\text{in}}) d\omega_{\text{in}}, \tag{3.23}$$

where H_S^- is the directional hemisphere containing all incoming directions ω_{in} towards the surface.

The radiance outgoing from the surface is composed of radiance emitted at s by the surface and all radiance scattered towards ω.

3.2.3 Appearance of the medium

The equation of transfer must answer a simple question. Without any scattering medium, the observer would basically perceive the radiance outgoing from the surface almost unchanged. With a scattering medium between s and r, how will radiance $\Gamma(s,\omega)$ be altered all way though the medium?

We already possess two key elements :

- The transport equation 3.21, which summarizes what happens to light going through an unitary phase volume. By summing all local contributions between s and r, we can obtain the global contribution of the presence of the medium in the scene. Because we plan to use the surface as a boundary to end the integration, we will start from r and end at s.
- Radiance $\Gamma(s,\omega)$ departing from the surface towards ω.

Path absorption function

The path absorption function computes the attenuation of radiance between two points x_a and x_b due to the presence of the scattering medium.

$$\beta(x_a,x_b) = e^{-\tau(x_a,x_b)}, \tag{3.24}$$

where $\tau(x_a,x_b)$ stands for the optical distance between x_a and x_b, i.e. the integral of the global interaction coefficient between the two points. These are dimensionless quantities.

The equation of transfer

The equation of transfer, expressing the total radiance arriving at the camera, can be written under this form :

$$L(r,\omega) = \beta(s,r)\Gamma(s,\omega) + \int_0^{\|r-s\|} \beta(r-x\omega,r) \times \left[\varepsilon(r-x\omega,\omega) + \int_{S^2} k(r-x\omega,\omega_{\text{in}},\omega)L(r-x\omega,\omega_{\text{in}})d\omega_{\text{in}}\right] dx \tag{3.25}$$

3.3 Common phase functions

3.3.1 Rayleigh and Mie scattering

The phase function expresses the angular distribution of light reflected inside a volumetric scattering entity such as the particles composing a gaseous medium, or a translucent matter in which light can penetrate and be scattered. This distribution is a function of the phase angle, indeed the angle between the incident direction and the reflected direction.

In a single-scattering medium, it is the angle between the direct light ray incoming on a particle, and the direction of the view ray, along which all in-scattered lighting is integrated. Because single-scattering only accounts for one bounce of light before it is redirected towards the camera, all radiance which is not redirected in this direction will never bounce again and therefore will not contribute to the final image.

Each phase function is associated to a type of light scattering. In our state of the art, we mention Rayleigh and Mie scattering, which both describe how light interacts with one different type of atmospheric particle. Some types of scattering can either keep the energy of the photons unchanged, what is called *elastic* scattering, or make it higher or lower, what is named *Raman* scattering.

The use of Rayleigh or Mie scattering depends on the size of the medium's particles in the real world. In atmospheric rendering, the common distinction is made between air molecules and aerosols on the one hand, and water droplets and pollution on the other hand. Air molecules and aerosols have a smaller size that the wavelength of the light (less than a tenth), this causes all wavelengths of the incoming sunlight to do not be scattered in the same proportions. Due to the small size of air molecules, smaller wavelengths are best scattered in the sky, and this is the reason why it appears blue. The interactions of light with larger spherical particles, like water in clouds, are best described by Mie scattering, for which all wavelengths are scattered in a more uniform fashion, making clouds appear white.

3.3.2 The Rayleigh phase function

Rayleight scattering is named after Lord John William Strutt Rayleigh (1842 - 1919), who won the Nobel Prize for Physics in 1904. In 1871, he was the first to formulate a possible explanation for the blue color of the sky, postulating that it might be related to the scattering of light rays on particles in the atmosphere. To base ourselves on a good explanation of how to obtain the Rayleigh phase function, we can turn to [REK+04].

We start from the expression of volume angular scattering coefficient $\rho(r, \alpha)$ for Rayleigh scattering :

$$\rho(r, \alpha) = \frac{\pi^2(n^2 - 1)}{2N\lambda^4} \left(1 + \cos^2\alpha\right), \tag{3.26}$$

where n is the refractive index of the air, N is the molecular number density of the air, and λ is the wavelength.

$\rho(r, \alpha)$ is the fraction of the incident radiance which is scattered in the direction forming an angle α with the incident direction.

To obtain the phase function, we need normalize $\rho(r, \alpha)$. We integrate over 4π which is the total solid angle over the complete sphere, to account for all directions, and obtain the total volume scattering coefficient $\rho(r)$:

$$\rho(r) = \int_0^{4\pi} \rho(r, \alpha)\ \mathrm{d}\alpha\ =\ \frac{8\pi^3(n^2 - 1)^2}{3N\lambda^4} \tag{3.27}$$

$\rho(r)$ represents the whole portion of incoming radiance which is scattered, and which by definition do not undergo any other behaviour than scattering. Therefore, $\rho(r)$ can be equal to 1, when light only undergoes scattering, or less than 1, as it is most likely, when other events than scattering can occur.

To obtain the normalized phase function $F(\alpha)$ for Rayleigh scattering, we divide $\rho(r, \alpha)$ by $\rho(r)$ to pass from the probability of an incoming ray to be scattered towards an angle α, to the probability of a ray, which we know will be scattered, to be deflected towards α :

$$F(\alpha) = \frac{\rho(r, \alpha)}{\rho(r)} = \frac{3}{16\pi}(1 + cos^2\alpha) \tag{3.28}$$

3.3.3 The Lorenz-Mie phase function

The Lorenz-Mie solution is a solution of Maxwell equations for the case of the scattering of light by spherical particles. It is named after Ludvig Lorenz (1829 - 1891), a Danish physicist and mathematician, and Gustav Mie (1859 - 1957), a German physicist.

The phase function for Mie scattering is given by :

$$F(\alpha) = \frac{1}{2}(|S_1|^2 + |S_2|^2), \tag{3.29}$$

where S_1 and S_2 are the *scattering amplitudes*, given by :

$$\begin{cases} S_1 = \displaystyle\sum_{n=1}^{+\infty} \frac{2n+1}{n(n+1)}(a_n\pi_n + b_n\tau_n) \\ S_2 = \displaystyle\sum_{n=1}^{+\infty} \frac{2n+1}{n(n+1)}(a_n\tau_n + b_n\pi_n) \end{cases} \tag{3.30}$$

The sequences $\{\pi_n\}$ and $\{\tau_n\}$ are defined in terms of the associated Legendre polynomial $P_n^1(\theta)$:

$$\begin{cases} \pi_n = \dfrac{P_n^1(\theta)}{\sin\,\theta} \\ \tau_n = \dfrac{\mathrm{d}P_n^1(\theta)}{\mathrm{d}\theta} \end{cases} \tag{3.31}$$

The sequences $\{a_n\}$ and $\{b_n\}$ are the *coefficients for Mie scattering*, expressed as :

$$\begin{cases} a_n = \dfrac{m\Psi_n(mx)\Psi_n'(x) - \Psi_n(x)\Psi_n'(mx)}{m\Psi_n(mx)\xi_n'(x) - \xi_n(x)\Psi_n'(mx)} \\ b_n = \dfrac{\Psi_n(mx)\Psi_n'(x) - m\Psi_n(x)\Psi_n'(mx)}{\Psi_n(mx)\xi_n'(x) - m\xi_n(x)\Psi_n'(mx)} \end{cases} \tag{3.32}$$

$\{\Psi_n\}$ and $\{\xi_n\}$ are Bessel - Ricatti functions.

Chapter 4

A brief introduction to wavelets

"How empty is theory in presence of fact!"

Mark Twain, *A Connecticut Yankee in King Arthur's Court*

4.1 Introduction

4.1.1 Overview

In this chapter, we quickly introduce the concepts behind wavelets and the wavelet decomposition, which are used in our work on real-time rendering of heterogeneous fog. Since the wavelet analysis is a comprehensive domain of both mathematics and signal processing of its own, we must stress that this is only a short introduction to wavelets, only focusing on the few elements that we really use in our work. For more information on wavelets, please refer to [SDS95] and [Mal08].

Wavelets are a mathematical tool to transform a single-layered signal into, on the one hand, a coarse resolution and, on the other hand, several layers of details. When all layers are summed together, the original signal can be recovered without any loss of information.

Wavelets appeared independently in engineering, physics and pure mathematics several decades ago, and since then have proved very popular in several other scientific domains. The underlying concepts are relatively simple to use, and did already find numerous applications in domains such as digital image processing, computer graphics, etc.

4.1.2 From the Fourier transform to multiresolution analysis

According to Fourier's theory, a signal can be expressed as a sum of potentially infinite series of sines and cosines, which is the popular Fourier series, i.e. which transposes into the frequency domain a signal expressed the standard way in the temporal domain. This representation gives a clear image of the whole set of frequencies which can be found in the signal, but leads to a complete loss of temporal informations. Indeed, although we have the information that certain frequencies are actually present inside the signal, we do not know exactly when. Talking about the precise

instant at which a frequency occurs makes no sense in itself, as a periodicity of a signal is something which can only be observed over a well-delimited lapse of time, or *period*. However, we can identify two temporal informations which can help describing a given frequency component, and which are missing in Fourier's frequency domain : the time intervals during which the frequency can be observed, and the phase associated to this periodic component at a given time t.

To deal with this problem, several solutions were put forward, in order to express a signal in both frequency and time domains. Nevertheless, those solutions were all based on the same idea : cutting the signal in sub-signals which can be analyzed separately. In simple words, in order to know when given frequencies can be observed, we have to study a smaller portion of the signal. If a frequency is observed in the Fourier transform of this smaller interval, although still no information regarding its phase is available, we have are more precise idea of its location.

The wavelet transform[1], based on this simple idea, is the most recent answer to this problematic of efficiently and systematically cutting a signal in smaller parts and recursively analysing them, thanks to the use of a sliding window. This window is moved classically along the whole length of the signal, and the spectrum is calculated again for each isolated portion. This process is then repeated a number of times while the window is reduced a bit more at each new iteration, together with the sliding step. The result is a collection of small spectra, obtained for different time intervals of the signal and for different precisions, we then speak of *multiresolution analysis*.

In this thesis, we only use the B-Spline wavelets, therefore for simplicity reasons we will only consider these wavelets in this chapter. In the next section, we explain how the wavelet framework is organized and enounce some general properties about one-dimensional wavelets. Then, we see how to perform Mallat's wavelet transform on a one-dimensional signal, where we first only consider Haar wavelets, i.e. the simplest type of wavelets. We continue by explaining the difference between 1D and 2D wavelets, and finally consider more sophisticated orders of B-Spline wavelets.

4.2 The wavelet framework

4.2.1 Data in a function basis

Similarly to a vector basis, which is built with orthogonal vectors chosen such that each vector cannot be expressed as a linear combination of the other vectors, the function bases that we use are built from a set of orthogonal functions F. This ensures that each signal s, which can be represented in such a function basis only admits one unique set of coefficients C in this basis, such that :

$$\forall x \in \mathscr{D}_s, s(x) = \sum_{k=0}^{n} c_k * f_k(x), \tag{4.1}$$

where $\{c_k\}$ are the coefficients associated to each basis function f_k, $\mathscr{D}_s$ is the definition domain of s, and n is the number of basis functions in the set F. If we continue our analogy with the vector basis, the set of coefficients $\{c_k\}$ can be viewed as the coordinates of the signal in the function basis.

[1]In this manuscript, we also refer to the wavelet transform under the term *wavelet decomposition*, which best describes this process.

Note that, in the general case, F may theorethically contain an infinite number of functions. In this chapter, for simplicity, we will consider n as finite.

4.2.2 Multiresolution analysis

4.2.2.1 The multiresolution pyramid

When we use wavelets, we represent data in a multiresolution pyramid, i.e. a pyramid of several function bases having a resolution evolving by a power of two between one level and the level below. The top of the pyramid, labelled level 0, represents data with the coarsest resolution. The bottom of the pyramid, labelled level $n-1$, has the highest resolution.

Each function basis is composed of orthogonal basis functions f_k, similar to an original pattern function f, which has been translated over the whole definition domain of the function basis to create all functions f_k. In the multiresolution pyramid, the pattern function f is first translated, to generate the basis functions of level 0, and then scaled by a factor of 2^{-n} to generate the basis functions of the $n-1$ lower levels.

4.2.2.2 Properties of basis functions for multiresolution analysis

We write $f_{j,k}$ the basis function translated by k steps at resolution level j :

$$f_{j,k}(x) = f(2^j x - k), \text{ with } j,k \in \mathbb{Z}, \tag{4.2}$$

where F is the space formed by the set of functions $\{f_{j,k}\}$, and F_j is the closed subspace of F containing all the elements that can be built in the function basis formed by the set of functions f of level j, written f_j.

We say that a function $f \in \mathbb{L}^2(\mathbb{R})$ creates a multiresolution analysis if it generates a sequence of nested closed subspaces F_j verifying the following properties :

1. $F_0 \subset F_1 \subset ... \subset F_{j-1} \subset F_j \subset F_{j+1} \subset ... \subset F_n$
2. $\bigcap_{j\in\mathbb{Z}} F_j = \{0\}$, and $clos_{\mathbb{L}^2}\left(\bigcup_{j\in\mathbb{Z}} F_j\right) = \mathbb{L}^2$
3. $f(x) \in F_j \Leftrightarrow f(2x) \in F_{j+1}, \text{ with } j \in \mathbb{Z}$
4. There exists ϕ such that $\phi(t-n), n \in \mathbb{Z}$ is an orthonormal basis of F_0; ϕ is called the scaling function, or father wavelet.

It can be difficult to imagine the relationship between these subspaces. The most important notion to understand is the fact that when we increase j by one, the resolution of the function basis increases by a power of two. In the multiresolution pyramid, we go down at the immediately lower level, the top of the pyramid corresponding to level 0. The more j increases, the more the subspace F_j generates elements. The subspace F_{j+1} generates twice more elements than F_j, indeed it generates all the elements that could be generated by F_j, plus one new element inserted between each two consecutive elements of F_j. In simple words, F_{j+1} is a function basis with twice the resolution of F_j.

4.2.3 The scaling function ϕ

4.2.3.1 Two types of function bases

Performing a wavelet decomposition on a signal actually involves the use of two types of function bases in which the different parts of the result will be expressed. A one-dimensional wavelet decomposition of a signal returns, first, a coarse approximation of the original signal and second, N layers of details generated along the N successive steps of decomposition. The coarse approximation is expressed in a *scaling functions* basis, and the details are the information expressed in wavelets, i.e. in N *wavelet functions* bases.

4.2.3.2 Defining ϕ

The scaling function, written ϕ, is completely distinct from the wavelet itself, even if wavelets are built from a combination of scaling functions. In the B-Spline wavelets family, its role is to bring all the energy of the non-decomposed original signal into the approximation so that, whatever the number of decomposition steps applied, i.e. however coarse the approximation is, the integral over its whole definition domain always corresponds to the integral over the original signal.

Multiresolution analysis requires the existence of a relationship linking two successive levels, in particular the ability to express the basis functions of a given resolution level as a function of the basis functions of the "lower" level in the multiresolution pyramid, which actually represent the informations with a higher resolution.

There exists no generic definition of the scaling function ϕ, since it must be known depending on the type of wavelets used. However, there exists a generic expression of the scaling function at level n as a function of thinner scaling functions at level $n+1$, called the *two-scale relationship* :

$$\phi(x) = \sum_{k=-\infty}^{\infty} p_k * \sqrt{2} * \phi(2x-k), \tag{4.3}$$

where $\{p_k\}$ are the coefficients of the *scaling sequence* of ϕ, which we will call *decomposition coefficients*. These coefficients must be known, and depend on the type of scaling function, and therefore on the wavelets used.

4.2.4 The wavelet function ψ

4.2.4.1 A little theory

Our subspaces F_j of basis functions verify the properties of multiresolution analysis, in particular the containment property such that $F_j \subset F_{j+1}$. We now define G_j as the complementary of F_j within F_{j+1} such that $F_{j+1} = F_j \oplus G_j$. We can write :

$$G_j \cup G_{j'} = \varnothing, \text{ with } j \neq j', \tag{4.4}$$

which means, in simple words, that although the subsets F_j are all mutually included the ones in the others, subsets G_j are mutually orthogonals, which is normal, since G_j are the elements added to F_j and which were missing to obtain F_{j+1}.

If J symbolizes the maximum resolution level, the set F_J is formed by the initial set F_0 to which we add all successive sets G_j :

$$F_J = F_0 \oplus \bigcup_{j=0}^{J-1} G_j \tag{4.5}$$

Functions $\{2^{j/2}\phi(2^j t - k, k \in \mathbb{Z}\}$ form an orthonormal basis of F_j, and functions $\{2^{j/2}\psi(2^j t - k, k \in \mathbb{Z}\}$ form an orthonormal basis of G_j.

The sets G_j also inherit the scaling property, such that :

$$f(x) \in G_j \Leftrightarrow f(2x) \in G_{j+1}, \text{ with } j \in \mathbb{Z} \tag{4.6}$$

Similarly to the set of functions $\{f_{j,k}\}$ which generated the subspace F_j, we now define the function g such that all $\{g_{j,k}\}$ generate the set G_j. The functions $g_{j,k}$ generate the details missing to F_j in order to double its resolution and obtain F_{j+1}.

If the functions $f_{j,k}$ are scaling functions, the functions $g_{j,k}$ correspond to the wavelets.

4.2.4.2 Defining ψ

The wavelet function at level j cannot be expressed in terms of thinner wavelets at the lower level $j+1$, like the scaling function does. There exists no relationship between the wavelet basis at level j and the wavelet basis at level $j+1$, whereas the *two-scale relationship* ensures this link between scaling functions bases of two successive resolutions.

Instead, we define the *two-scale relationship for wavelets*, which makes the link between a wavelet function at level j and scaling functions at level $j+1$:

$$\psi(x) = \sum_{k=-\infty}^{\infty} q_k * \sqrt{2} * \phi(2x - k), \tag{4.7}$$

where $\{q_k\}$ are the coefficients of the scaling sequence of ψ, which we call the *wavelet coefficients*.

Since F_{j+1} includes all the elements generated by functions $\{f_{j,k}\}$ from F_j, and $\{g_{j,k}\}$ from G_j, it is possible to express functions $\{f_{j,k}\}$ in terms of $\{f_{j+1,k}\}$, it is however impossible to express functions $\{g_{j,k}\}$ in terms of $\{g_{j+1,k}\}$ since G_{j+1} is the complementary of F_{j+1} and thus cannot, by definition, contain any element of G_j.

For this reason, there exists no sequence of coefficients $\{p_k\}$ such that : $\psi(x) = \sum_{k=-\infty}^{\infty} p_k * \psi(2x - k)$.

4.3 Haar and B-Spline wavelets

4.3.1 Haar wavelets

Haar wavelets are the simplest type of wavelets, very convenient for a first approach with the mechanisms behind multi-resolution.

4.3.1.1 Haar scaling function

Haar scaling function, also known as the Haar function, is defined on $[0, 1]$ by :

$$\phi(x) = \begin{cases} 1 \text{ for } 0 \leq x < 1 \\ 0 \text{ otherwise} \end{cases} \tag{4.8}$$

Its coefficients of decomposition are $p_0 = 1$ and $p_1 = 1$, which gives the following scaling relation :

$$\phi(x) = \phi(2x) + \phi(2x+1) \tag{4.9}$$

4.3.1.2 Haar wavelet function

The Haar wavelet is defined by the two coefficients $q_0 = 1$ and $q_1 = -1$, which gives :

$$\psi(x) = \phi(2x) - \phi(2x-1) \tag{4.10}$$

$$\psi(x) = \begin{cases} 1 \text{ for } 0 \leq x < 0.5 \\ -1 \text{ for } 0.5 \leq x < 1 \\ 0 \text{ othsewise} \end{cases} \tag{4.11}$$

4.3.2 B-Spline wavelets

4.3.2.1 B-Spline scaling functions

B-Spline functions, which we will write N, are defined for different orders[2] $m \geq 2$, corresponding to degrees $m-1$ of B-Splines. A B-Spline function of order m is defined by :

$$N_1(x) = \begin{cases} 1 \text{ for } 0 \leq x < 1 \\ 0 \text{ otherwise} \end{cases} \tag{4.12}$$

$$N_m(x) = \int_{-\infty}^{\infty} N_{m-1}(t) N_1(t) dt \tag{4.13}$$

To be easily implemented, a B-Spline of order m can also be defined under the more convenient form :

$$N_m(x) = \frac{x}{m-1} N_{m-1}(x) + \frac{m-x}{m-1} N_{m-1}(x-1), \tag{4.14}$$

A B-Spline of order m is defined on the following domain :

$$\mathscr{D}_{N_m} = [0, m] \tag{4.15}$$

Its subdivision coefficients are given by :

$$p_k = 2^{-m+1} \binom{m}{k}, \text{ with } 0 \leq k < m \tag{4.16}$$

[2]The B-Spline of order 1 (degree 0) is the Haar function.

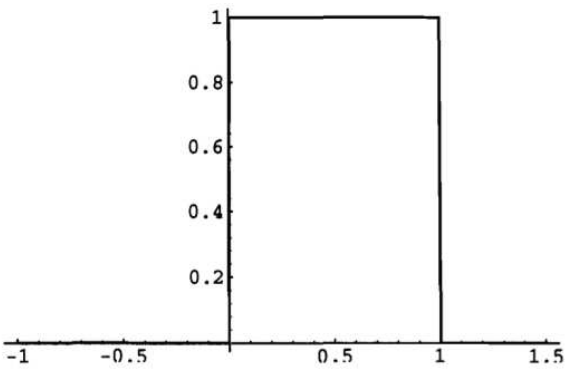

Figure 4.1: The Haar (scaling) function.

And its two-scale relationship is the following :

$$N_m(x) = \sum_{k=0}^{m} p_k N_m(2x - k) \tag{4.17}$$

4.3.2.2 B-Spline wavelet function

The wavelet coefficients of a B-Spline of generic order are given by :

$$q_k = (-1)^k 2^{1-m} \sum_{l=0}^{m} \binom{m}{l} N_{2m}(k + 1 - l) \tag{4.18}$$

Its two-scale relationship for wavelets is as follows :

$$\psi_m(x) = \sum_{k=0}^{3m-2} q_k N_m(2x - k) \tag{4.19}$$

The B-Spline wavelet is defined on the domain :

$$\mathscr{D}_{N_m} = [0, 2m - 1] \tag{4.20}$$

4.4 The wavelet transform in one dimension

Note : There exists several wavelet transform algorithms, the main methods being the continuous wavelet transform, the discrete wavelet transform, and Mallat's fast wavelet transform. In this thesis, we use Mallat's algorithm, which is the simplest to implement and the most adaptated to the decomposition of discrete data such as images.

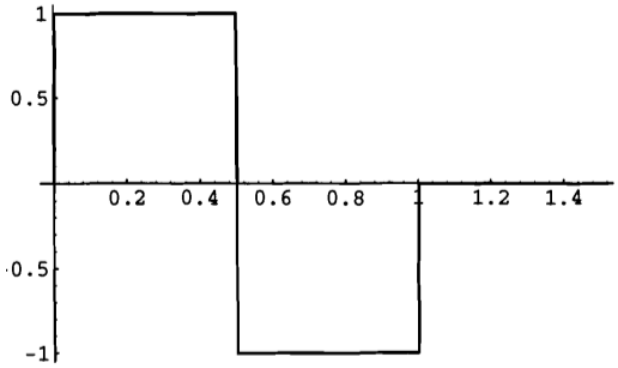

Figure 4.2: The Haar wavelet function.

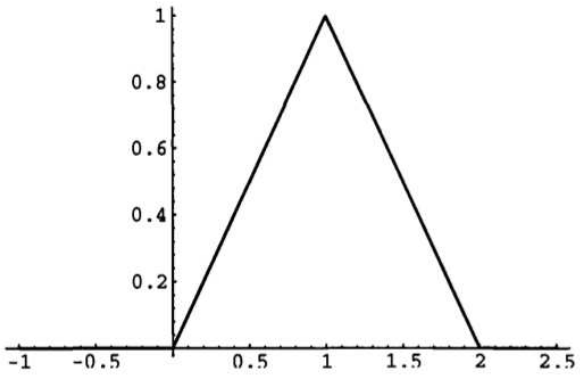

Figure 4.3: The linear B-Spline (scaling) function.

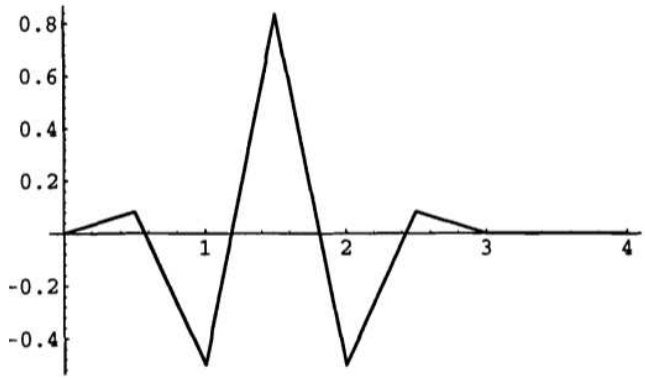

Figure 4.4: The linear B-Spline wavelet function.

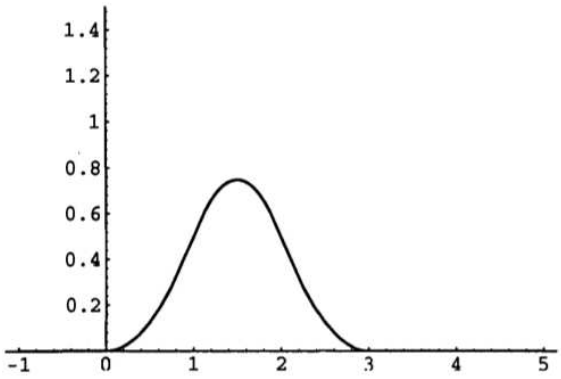

Figure 4.5: The quadratic B-Spline (scaling) function.

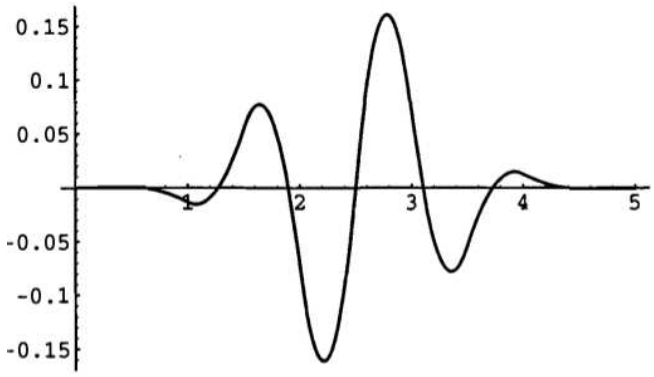

Figure 4.6: The quadratic B-Spline wavelet function.

4.4.1 The idea

In section 4.2.2, we saw how to build the scaling function ϕ and the wavelet ψ, the two pattern basis functions. The scaling functions and wavelets basis are created by translating these pattern functions, and the different levels of the multiresolution pyramid are obtained by scaling these basis functions by a power of two.

From this point, the algorithm of the decomposition is fairly simple. The data provided as input can be any 1D, 2D or 3D discrete data, but for simplicity reasons we chose to first stay in one dimension, and consider higher dimensions later.

When one decomposition step is performed on a sequence of m values, two subsequences of $m/2$ values are obtained :

- An approximation of the signal at half its resolution.
- The details which were extracted from the signal, and which are now missing in the approximation to recover the original data.

One element is important to understand : both the signal in input and the decomposed data in output are not directly manipulated by their discrete samples, but are all modeled in function bases.

- The values given to the decomposition algorithm are m coefficients modeling the signal in a **scaling functions** basis, at level j.
- The decomposition generates two sets of values :
  - $m/2$ coefficients modeling the approximation in a **scaling functions** basis, at level $j-1$ which has half the resolution of level j.
  - $m/2$ coefficients modeling the extracted details in a **wavelet functions** basis, also at level $j-1$.

Without the use of wavelets, such a decomposition would still be easy to perform, with the difference that the details extracted from the signal at level j to generate the approximation at level $j-1$ would, by definition, be modeled with the same resolution of the signal, i.e. at level j. Because the wavelet at level j has a shape which is actually twice thinner than the resolution of the scaling function of this level, both the approximation and its removed details can be modeled using respectively scaling and wavelet functions bases of level $j-1$. Where this aspect is interesting is that the total number of coefficients obtained after a large number of decomposition steps remains the same as the number of values expressing the original signal.

4.4.2 Decomposition and reconstruction relationships

We consider again the space F formed by the set of functions $\{f_{j,k}\}$, and F_j the closed subspace containing all the elements that can be created in the function basis formed by all functions f of level j, written f_j. The space G_j is the complementary of F_j in F_{j+1} such that $F_{j+1} = F_j \cup G_j$.

The two-scale relationships of the scaling function (equation 4.3) and the wavelet (equation 4.7) are called *reconstruction relationships*.

Since both $\phi(2x)$ and $\phi(2x-1)$ belong to F_1 and $F_1 = F_0 \oplus G_0$, we can define four sequences of coefficients $\{a_{-2k}\}$, $\{b_{-2k}\}$, $\{a_{1-2k}\}$ and $\{b_{1-2k}\}$, with $k \in \mathbb{Z}$ such that :

$$\phi(2x) = \sum_k [a_{-2k}\phi(x-k) + b_{-2k}\psi(x-k)] \tag{4.21}$$

$$\phi(2x-1) = \sum_k [a_{1-2k}\phi(x-k) + b_{1-2k}\psi(x-k)] \tag{4.22}$$

which gives us :

$$\phi(2x-l) = \sum_k [a_{l-2k}\phi(x-k) + b_{l-2k}\psi(x-k)],\ l \in \mathbb{Z} \tag{4.23}$$

This equality is called *decomposition relationship* for ϕ et ψ. The sequences of coefficients $\{a_k\}$ and $\{b_k\}$ are called *decomposition sequences* and are crucial for the wavelet decomposition algorithm.

4.4.3 Decomposition algorithm

According to the general structure of multiresolution analysis, the set F_j of functions f_j at a given subdivision level j is created by all possible translations of a scaling function $\phi_j \in L^2(\mathbb{R})$, and the set of functions g_j is created by translating the associated wavelet function $\psi_j \in L^2(\mathbb{R})$. According to the property of completeness, which states that the fusion of all sets F_j constitutes the whole set $L^2(\mathbb{R})$, each function f can be approximated as precisely as possible by a function $f_N \in F_N$, for any $N \in \mathbb{Z}$. Since $F_j = F_{j-1} \oplus G_{j-1}$ for any $j \in \mathbb{Z}$, indeed the approximation at one given level j is formed by the coarser approximation at the lower level $j-1$ combined with the details at $j-1$, we deduce that f_N admits an unique decomposition :

$$f_N = f_{N-1} + g_{N-1} \tag{4.24}$$

with $f_{N-1} \in F_{N-1}$ and $g_{N-1} \in G_{N-1}$.
By repeating this process, we manage to obtain :

$$f_N = g_{N-1} + g_{N-2} + ... + g_{N-M} + f_{N-M} \tag{4.25}$$

with $f_j \in F_j$ and $g_j \in G_j$.
This way, we recover the equation of the wavelet decomposition.
We can notice that $f_j \in F_j$ and $g_j \in G_j$ both admit a representation in unique series :

$$\begin{cases} f_j(x) = \sum_k c_{j,k}\phi(2^j x - k), \\ \qquad \text{with } c_j = \{c_{j,k}\} \in L^2 \end{cases} \tag{4.26}$$

$$\begin{cases} g_j(x) = \sum_k d_{j,k}\psi(2^j x - k), \\ \qquad \text{with } d_j = \{d_{j,k}\} \in L^2 \end{cases} \tag{4.27}$$

More precisely, each individual function $f_j \in F_j$ can be expressed in an exact fashion by the sum of individual products of each coefficient $c_{j,k}$ with the scaling function $\phi(2^j x - k)$, a version of the pattern function ϕ at level j and translated at $k \in \mathbb{Z}$. Similarly, each individual function $g_j \in G_j$ can be expressed exactly by the sum of the products of each $d_{j,k}$ with the wavelet $\psi(2^j x - k)$.

As you may have guessed, the coefficients $c_{j,k}$ model, in the scaling functions basis F_j, the approximation generated by the decomposition, and the coefficients $d_{j,k}$ model the details extracted, in the wavelets basis G_j.

The main equations of the decomposition algorithm are as follows :

$$\begin{cases} c_{j-1,k} = \sum_l a_{l-2k} c_{j,l} \\ d_{j-1,k} = \sum_l b_{l-2k} c_{j,l} \end{cases} \tag{4.28}$$

Let us consider an one-dimensional signal composed of $n = 32$ samples, from which we would like to extract the layers of details using the wavelet decomposition. Because each step divides the resolution of the approximation by a factor of two, we know that the maximum number of decompositions that can be performed on this signal is : $ln(n)/ln(2) = 5$.

The signal should normally first be expressed in a scaling functions basis before the wavelet decomposition can be applied, however if we use Haar wavelets, the signal samples directly correspond to their own coefficients in the Haar scaling function basis. Haar wavelets are the most well-known, simple and commonly used wavelets, especially for decomposing images, since the pixel values can directly be passed to the decomposition algorithm.

Initially, the approximation pyramid only contains the coefficients of the original signal at level $j = 5$, as the thinnest approximation possible, and the details pyramid is empty, since no layer of details was extracted yet.

Applying the first step of wavelet decomposition involves using equations (4.28) to compute the sequence of scaling function coefficients $\{c_{j-1,k}\}$ for the approximation at level $j - 1 = 4$, and the sequence of wavelet basis coefficients $\{d_{j-1,k}\}$ for the associated details, in a wavelet basis at level 4. The sequences $\{a_{l-2k}\}$ and $\{b_{l-2k}\}$ are know according to the type of wavelets. In the case of Haar wavelets, although the scaling basis coefficients $\{c_{j-1,k}\}$ directly represent the samples of the approximation, this is not true for the details coefficients $\{d_{j-1,k}\}$ which must be multiplied by the Haar wavelet function for a correct evaluation. The 16 values of the new approximation $\{c_{4,k}\}$ are pushed on top of the approximations pyramid, and the 16 coefficients of details $\{d_{4,k}\}$ are pushed on top of the details pyramid, both at level $j = 4$.

This process is repeated by taking as input the latest approximation $\{c_{4,k}\}$, which generates the 8 coefficients of an even coarser approximation $\{c_{3,k}\}$ with a new layer of details $\{d_{3,k}\}$, and the algorithm continues recursively until the approximation and details at level 0 are obtained. The approximation at level 0 only contains one single value, it therefore cannot be decomposed any further.

Of the whole process it results :

- 1 very coarse one-pixel approximation $c_{0,0}$.
- 1 coefficient of details $d_{0,0}$ (5^{th} layer), in the wavelets basis G_0.
- 2 coefficients of details $\{d_{1,0}, d_{1,1}\}$ (4^{th} layer), in the wavelets basis G_1.

- 4 coefficients of details $\{d_{2,0}, ..., d_{2,3}\}$ (3^{th} layer), in the wavelets basis G_2.
- 8 coefficients of details $\{d_{3,0}, ..., d_{3,7}\}$ (2^{nd} layer), in the wavelets basis G_3.
- 16 coefficients of details $\{d_{4,0}, ..., d_{4,15}\}$ (1^{st} layer), in the wavelets basis G_4.

Without the use of wavelets, performing the same decomposition by simple downsampling and averaging would have produced $1 + 2 + 4 + 8 + 16 + 32 = 63$ values. With Haar wavelets, separating the frequencies of a signal of 32 samples produces no more than 32 coefficients, which is an interesting feature in terms of memory cost when working, for example, on large 2D images.

When the approximation is summed with all the layers of details, the exact original signal $s(x) = s_N(x)$ is recovered.

In the general case, the approximation at level i can be recovered as follows :

$$s_i(x) = \sum_{k=-\infty}^{\infty} c_{0,k}\phi_{0,k}(x) + \sum_{j=0}^{i-1}\sum_{l=-\infty}^{\infty} d_{j,l}\psi_{j,l}(x) \tag{4.29}$$

4.4.4 Some decomposition sequences

4.4.4.1 Role of the decomposition sequences $\{a_k\}$ and $\{b_k\}$

We have just seen the decomposition algorithm with the two key relations (4.28) allowing to decompose a signal into a coarser approximation and a layer of details. The only elements that we are missing for these equations to be really exploitable are the two decomposition sequences $\{a_k\}$ and $\{b_k\}$, different for each type of wavelet.

To extract from the signal each single coefficient at a position $x = k$ on the approximation and on the layer of details, equation 4.28 involves computing a linear combination of the coefficients in a neighbourhood around the position $x = k$ on the input signal. This mechanism is similar to a convolution, where the sequences $\{a_k\}$ and $\{b_k\}$ play the role of the filters to extract the coefficients of respectively the next approximation and its associated details. Note that the sum of all a_k and b_k always equals 1.

4.4.4.2 $\{a_k\}$ and $\{b_k\}$ for Haar wavelets

The decomposition sequences for Haar wavelets are relatively simple. Computing the approximation at level $j - 1$ simply means averaging the signal's samples 2 by 2. For Haar wavelets, we have :

$$\begin{cases} \{a_k\} = \left\{\frac{1}{2}; \frac{1}{2}\right\} \\ \{b_k\} = \left\{\frac{1}{2}; -\frac{1}{2}\right\} \end{cases} \tag{4.30}$$

4.4.4.3 $\{a_k\}$ and $\{b_k\}$ for B-Spline wavelets

With B-Spline wavelets, things get a little more sophisticated as the scaling and wavelet basis functions are defined on a domain larger than 1, which makes two neighbouring basis functions overlay each other. The convolution filter actually has the same length as the signal itself. The generic decomposition sequences for the B-Splines of order m is :

$$\begin{cases} a_k = \frac{1}{2}(-1)^{k+1} \sum_{l \in \mathbb{Z}} q_{-k+2m-1-2l} c_{l,2m} \\ b_k = -\frac{1}{2}(-1)^{k+1} \sum_{l \in \mathbb{Z}} p_{-k+2m-1-2l} c_{l,2m} \end{cases} \tag{4.31}$$

All the difficulty now lies in the computation of coefficients $\{c_{k,m}\}$, as follows :

$$\sum_k c_{k,m} z^k = \frac{1}{\sum_k N_m \left(k + \frac{m}{2}\right) z^k}, \tag{4.32}$$

where $z = e^{\frac{-ix}{2}}$.
For linear B-Splines ($m = 2$), we obtain this equation :

$$c_{l,4} = (-1)^l \sqrt{3}(2 - \sqrt{3})^{|l|} \tag{4.33}$$

With quadratic B-Splines ($m = 3$), we gain even more in complexity :

$$c_{l,6} = \frac{120}{\alpha_1 - \alpha_2} \left(\frac{\beta_1^{|l|}}{\beta_1 - \beta_2} - \frac{\gamma_1^{|l|}}{\gamma_1 - \gamma_2} \right) \tag{4.34}$$

with :

$$\begin{cases} \alpha_1 = -13 + \sqrt{105} \\ \alpha_2 = -13 - \sqrt{105} \end{cases} \tag{4.35}$$

$$\begin{cases} \beta_1 = \frac{1}{2}(\alpha_1 + \sqrt{\alpha_1^2 - 4}) \\ \beta_2 = \frac{1}{2}(\alpha_1 - \sqrt{\alpha_1^2 - 4}) \end{cases} \tag{4.36}$$

$$\begin{cases} \gamma_1 = \frac{1}{2}(\alpha_2 + \sqrt{\alpha_2^2 - 4}) \\ \gamma_2 = \frac{1}{2}(\alpha_2 - \sqrt{\alpha_2^2 - 4}) \end{cases} \tag{4.37}$$

4.5 Wavelets in two dimensions

4.5.1 2D wavelets as a tensor product of two 1D functions

Adapting the scaling functions and wavelets for two dimensions consists in assigning a 1D function to each axis x and y, and evaluating the product of these two 1D functions. It is therefore possible

to create all possible combinations by assigning a different type of function to each axis to create hybrid scaling-wavelet functions. The 2D decomposition algorithm actually involves the use of all four combinations of functions that can be created.

Similarly, adaptating the function bases in 2D simply consists in multiplying the two 1D coefficients corresponding to the coordinate on each axis, to obtain the corresponding 2D coefficient.

However, the wavelet decomposition in 2D **does not** just consists in multiplying, to compute a value, the coefficients a_k and b_k on each axis to obtain their 2D counterpart. There actually does not exists a true 2D decomposition algorithm, but only a combination of successive 1D decompositions on each axis.

4.5.2 Algorithm

The wavelet decomposition in two dimensions is achieved, for example on an image, by decomposing separately the rows and the columns. We distinguish two approaches : the *standard* and the *nonstandard* decomposition.

The nonstandard decomposition (figure 4.7) consists, first, in decomposing completely all rows of all levels, which generates a list of separate 1D decompositions, each decomposition being composed of an approximation and several layers of details. Then, all the columns are decomposed on all levels on the approximation and on each type of details obtained after the decomposition on rows. This implies that all lines corresponding either to the approximation, either to the same layer of details have first to be assembled together.

The standard decomposition (figure 4.8), much more convenient to implement, consists in alternatively decomposing the rows then the columns, indeed each level is completely processed at each step.

Nevertheless, the two types of decomposition generate the same result :

- A coarse approximation at the root level $j = 0$, modeled in a basis of 2D scaling-scaling functions defined by $\phi\phi(x, y) = \phi(x)\phi(y)$.

- The vertical details of the horizontal approximation, in several layers, modeled in a basis of 2D scaling-wavelet functions defined by $\phi\psi(x, y) = \phi(x)\psi(y)$.

- The vertical approximation of the horizontal details, in several layers, modeled in a basis of 2D wavelet-scaling functions defined by $\psi\phi(x, y) = \psi(x)\phi(y)$.

- The vertical details of the horizontal details, in several layers, modeled in a basis of 2D wavelet-wavelet functions defined by $\psi\psi(x, y) = \psi(x)\psi(y)$.

In the general case, in two dimensions, the approximation at level i can be recovered as follows :

$$
\begin{aligned}
s_i(x,y) = & \sum_{k_x=-\infty}^{\infty} \sum_{k_y=-\infty}^{\infty} c_{0,k_x,k_y} \phi\phi_{0,k_x,k_y}(x,y) \\
& + \sum_{j=0}^{i-1} \left(\sum_{l_x=-\infty}^{\infty} \sum_{l_y=-\infty}^{\infty} d^{\phi\psi}_{j,l_x,l_y} \phi\psi_{j,k_x,k_y}(x,y) \right. \\
& \qquad + \sum_{l_x=-\infty}^{\infty} \sum_{l_y=-\infty}^{\infty} d^{\psi\phi}_{j,l_x,l_y} \psi\phi_{j,k_x,k_y}(x,y) \\
& \qquad \left. + \sum_{l_x=-\infty}^{\infty} \sum_{l_y=-\infty}^{\infty} d^{\psi\psi}_{j,l_x,l_y} \psi\psi_{j,k_x,k_y}(x,y) \right)
\end{aligned}
\tag{4.38}
$$

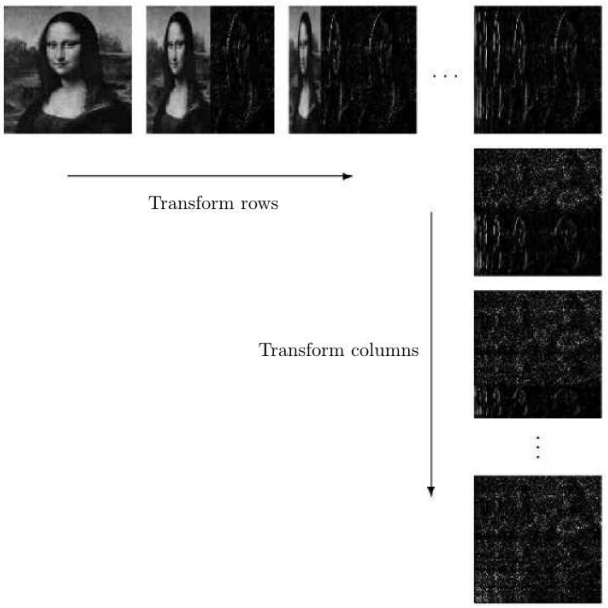

Figure 4.7: Overview of the nonstandard wavelet decomposition algorithm.

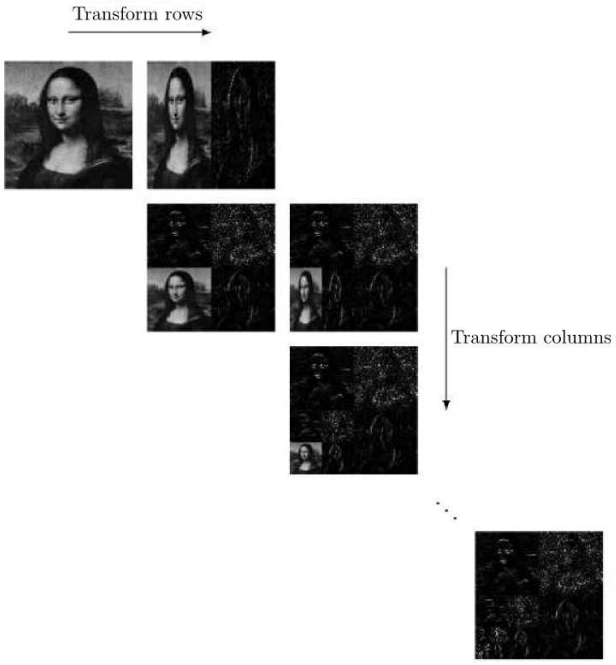

Figure 4.8: Overview of the standard wavelet decomposition algorithm.

Part II

Review of literature

Chapter 5

Categories of rendering methods

"The moon gazed on my midnight labours, while, with unrelaxed and breathless eagerness, I pursued nature to her hiding places."

Mary Shelley, *Frankenstein*

5.1 The problematic : from equations to pixels

Note : For a more extensive review of early techniques, we invite the reader to read the excellent survey by Cerezo et al. [CPCP+05].

5.1.1 Theoretical basis : the equation of transfer

In the previous part, we have defined the notion of a participating medium. We first established step-by-step the transport equation, which is the relationship between the quantity of photons entering and the quantity of photons exiting an unit volume inside a scattering medium. The transport equation expresses explicitely the different possible behaviours of light very locally, at each point inside the medium, and is not yet related in any extend to the global appearance of the medium.

In a second time, we established the equation of transfer, which expresses the total radiance arriving at a position inside or outside the medium, and from a specified direction. This second equation directly defines the global appearance of the medium as seen from an observer's position and according to his viewing direction. It simply consists in the sum of the radiance due to elements other than the medium, generally solid surfaces, and the global contribution of the medium in the scene, obtained by integrating the transport equation along the view ray.

5.1.2 Obtaining an image : the need for an approximation model

To compute an image based on the content of a scene, one must determine the radiance reaching each pixel on the camera plane. This involves solving the equation of transfer as a function of both the observer's position and the view direction.

Except for very simple situations, when objects and lighting reproduce conditions similar to the real world, this is absolutely not a trivial process, for the following reasons :

- The equation of transfer features a first integral along the view ray. An exact solution can be computed only if the distribution of the medium's density can be fully expressed as a mathematical function.

  When the medium is arbitrarily heterogeneous, i.e. when the density function is discretely sampled, it is not possible to come with a more factorized mathematical definition other than the simple explicit sum over all samples. In this case, a numerical integration based on a ray-marching along the view ray is required, which can be costly in terms of performance.

- Moreover, this first integral is recursive, since the radiance in-scattered at each sampled position along the view ray involves a nested second integral summing all radiance incoming at this position from all directions, whether direct or already scattered. Whatever the situation and without any limit on the scattering order, this second integral is impossible to solve analytically, for the simple reason that the light incoming from each direction is itself the result of a recursive call on the same equation. In the real world, although a photon cannot bounce infinitely on the medium's particles, there is no natural obvious and fixed limit to the recursivity of the equation of transfer.

Because for these reasons the equation of transfer is impossible to evaluate in its original mathematical form, in order to compute a pixel color, derivating a simplified model from the original equation of transfer cannot be avoided.

5.2 Types of rendering methods

5.2.1 Different approaches

All algorithms trying to simulate the illumination of solid objects or participating media by one or more light sources possess the same ultimate goal : solving the equation of transfer and computing a color for each pixel. However, depending on the type of application and the available hardware capabilities, their priorities are different, and so are the nature and quantity of adaptations performed on the original model.

We can distinguish two completely different approaches for integrating the distribution of light in the scene :

- **Deterministic approaches :** The result is fully determined by all parameters of the scene (point of view, view direction, scene content, lights, materials, etc.), therefore the same parameters always lead to the same result. The original equations are approximated by simplifying the behavior of light. As an example, single-scattering models remove the recursivity of the equation of transfer by limiting to only one the number of possible bounces of light rays on the medium before they reach the camera. A deterministic approximation consists in computing a nearly exact result for a model after it was very simplified to make this possible.

- **Stochastic approaches :** Computing the distribution of light in the scene involves random processes such as Monte-Carlo integration and random sampling, therefore two consecutive

renderings with the same scene and the same parameters may lead to slightly different pixel colors, due to a different noise. Here, even if very few or no simplifications are made on the original model to keep the behavior of light as close to reality as possible, an approximation still lies in the limited number of paths that are traced through the scene.

Historically, however, participating media rendering algorithms have usually been classified in three categories : analytical, deterministic and stochastic methods.

5.2.2 Analytical methods

A special type of deterministic method, where the integral of the radiance arriving at a pixel from the view direction is directly expressed using a comprehensive equation taking into account all parameters which may affect the result, such as the oberver's position and view direction. An analytical method is only possible when a mathematical formula can be found which characterizes the visual contribution of each component of the scene to the final color of pixels.

Most analytical methods are applied on homogeneous or layered media, for example in atmospheric rendering. Deriving an analytical formulation of an illumination model often requires a lot of assumptions and strong simplifications in an attempt to reduce the number of parameters of which the final pixel value depends. This can enable more precomputations and make real-time reachable.

5.2.3 Deterministic methods

Most modern real-time algorithms are deterministic, indeed they do not appeal to random processes, but cannot be labelled as analytic because instead of being defined using a mathematical equation, participating media or other elements are modeled using a discrete set of samples, with arbitrary used-defined values. This is analogous to the difference between vector and discrete graphics.

Typical examples are media modeled using a regular volumetric grid of voxels, or defined using a list of spherical particles such as radial basis functions (RBF). In comparison with analytical methods, pure deterministic algorithms use numerical integrations, e.g. they account for all medium samples situated in front of a pixel by performing a discrete ray-marching along the view ray. Because of the lack of an equation which would characterize the medium, it is impossible to have an idea of its shape without passing through all samples which contribute to the image.

Working with samples involves heavier computations than with mathematically defined media, but allows for more complex and more realistic shapes, while keeping real-time possible through the use of graphics hardware.

5.2.4 Stochastic methods

Monte-Carlo integration proves particularly useful for quickly approximating the integral of a multi-dimensional function over a finite domain, by randomly picking points within this domain and summing their respective images by the function considered.

A typical application of Monte-Carlo integration for illumination purposes is Monte-Carlo ray-tracing, a stochastic adaptation of ray-tracing where a large number of behaviors (an infinity, for

diffuse reflections) are possible when a ray hits a surface or interacts with a medium, leading to an exponential increase of the number of possible paths. In this case, considering all possible paths that photons can take would be astronomically costly. Instead, a finite number of rays are launched from each single pixel and are traced through the scene where they all randomly take a different path, depending on probabilities associated to the material properties of the objects. Their contributions are then averaged to construct the final pixel color.

Monte-Carlo integration requires a high number of samples for the result to converge and become stable. Although very noisy images can be produced very quickly, at least several thousands of rays per pixel are required to obtain noise-free images, which has always made stochastic methods unsuitable for real-time.

5.2.5 Conclusion

In the next sections of this quick survey, we only discuss techniques that may help reaching real-time, considering that this is an important aspect of our work.

Methods based on a discretization of light into photons or using a Monte-Carlo integration for solving the distribution of radiance in the scene are too far from real-time and will not be described. Non-stochastic methods based on radiosity are too expensive, especially when volumes are introduced in the equations. Even for computing one radiance scattering step between visible elements, a lot of neglectable transfers are computed which could be spared.

For this reason, our review of existing techniques will mainly focus on deterministic methods and the most interesting analytical methods, the category which a lot of early historical works belong to. Moreover, while some methods related to our work consider coarse approximations of multiple-scattering, we clearly emphasize on single-scattering illumination.

In the next chapter, we compare different ways of modeling participating media by discussing old techniques, i.e. which do not use hardware acceleration. Finally, in a third chapter, we review recent real-time scattering media illumination methods and discuss their strategy to exploit the capabilities of graphics hardware to reach real-time as efficiently as possible.

Chapter 6

1980 - 2000 : Modeling and rendering participating media

"The human brain is capable of only one strong emotion at a time, and if it be filled with curiosity or scientific enthusiasm, there is no room for fear."

Sir Arthur Conan Doyle, *The Brown Hand*

In this chapter, we present some of the most important historical participating media rendering methods, published during the years 1980s and 1990s, before the arrival of mainstream graphics hardware. This allows us to emphasize on discussing how the medium is modeled regarding the type of application.

6.1 Analytical techniques based on homogeneous layers

6.1.1 Introduction

6.1.1.1 Overview

When the interest in participating media rendering began in the early 1980s, real-time simulations were not conceivable. Due to the absence of graphics hardware enabling natively pre-integrated features such as hardware texturing and blending as well as strong memory limitations, conceiving a fully generic volume rendering method which would handle any kind of medium and render all visual effects whatever the type of scene was barely an option, so putting a number of constraints on the medium and the scene could not be avoided.

Nevertheless, deciding between homogeneous or heterogeneous media was not just a matter of choosing between simplicity and complexity. The type of medium and the rendering strategies adopted by early works were driven by the precise type of effect that the researchers wanted to simulate, in a nutshell, the object of the simulation.

The first works show a clear distinction between :

- The simulation of lighting effects, such as volumetric beams of light between clouds or atmospheric scattering, for which homogeneous or analytically defined media are best adaptated. In the real world, such effects occur when light passes through a medium with an almost constant or softly changing density which makes its presence barely visible, unless such illumination effects reveal its presence. In this case, the distribution of lit and shadowed zones is not caused by the low frequency medium itself, but by the higher frequency occluders such as solid geometry (e.g. a window letting sunlight enter in a dusty room, or trees in a forest) or smaller and denser heterogeneous media like clouds.

- The visualization of the medium itself as the main *object* of the application, which is generally heterogeneous and mush also be delimited spatially in order to be clearly identified. Although the first category always work with scenes including both a scattering medium and solid geometry, it is not rare that some volume visualization applications only consider volumes without solid surfaces, sometimes because the algorithm is generic enough to handle volumetric data representing both gaseous and solid objects.

6.1.1.2 About our classification

In this first section, we focus on homogeneous media, for which all methods employ analytical models. This is not a surprise, since the possibility of an analytical model is the biggest advantage of the use of a homogeneous medium in comparison with a heterogeneous one. We chose to classify those works in three categories, according to the type of scene :

- Homogeneous media in generic scenes rendered using scanlines and shadow volumes. In these methods, the medium is modeled using a single constant density value, and is assumed to simply spread over the whole scene.

- Simulation of the scattering of sunlight through the atmosphere. The medium does not cover the whole scene, but is modeled under the form of several layers of constant density, delimited vertically using minimum and maximum altitude boundaries. Some heterogeneous clouds may be added in the scene, but are modeled separately.

- Underwater scenes, simulating the interaction of sunlight with water, modeled as a homogeneous medium delimited by a mesh defining the surface of the pool or the sea.

To begin, it can be interesting to recall one of the most well-known historical methods [Bli82b], very specific in comparison with other works to come.

6.1.1.3 Back in 1982 : J. F. Blinn illuminates the rings of Saturn

Overview Development and implementation of participating media rendering really began with this work [Bli82b], which aims at simulating the reflection of light over a layer of spherical particles, and which is applied to the very specific application of visualizing of the rings of planet Saturn. At the time, simple local illumination models like Lambert or Phong were already available, but no such model was adapted to the simulation of fuzzy or dusty surfaces.

Blinn starts with a complete theoretical illumination model, and applies several majors simplifications. He finally obtains a completely analytical model, indeed for which the radiance reaching

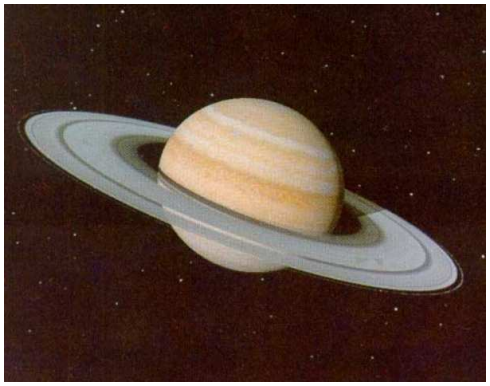

Figure 6.1: Saturn surrounded by its rings of dust, illuminated by single-scattering of sunlight. By Blinn [Bli82b].

the viewer is obtained without any numerical integration or ray-marching but instead by evaluating a single equation parameterized by all needed parameters.

Description First, only single-scattering is simulated, and all particles are assumed to be identical in radius.

Then, both the light source and the viewer are considered at infinity, this way the observer and the light both cannot lie inside the medium itself. This assumption ensures that the medium is always visualized from a large distance. This would be a heavy constraint for most of today's real-time applications, but is essential here, so that it makes sense to do not actually compute the contribution of each single particle of dust, and only consider an overall local approximation. Indeed, instead of having each single particle bringing its full contribution or not, Blinn uses a statistical approach and approximates the overall brightness of the particles cloud by a Poisson process, representing the local probability for a particle to do not be occluded from light rays.

The radiance reflected towards the camera only vary according to the phase angle, i.e. the angle between the incident lighting direction and the direction from the particle to the viewer, and the actual brightness reflected towards the viewer will be given by the phase function.

Several different phase functions are discussed in the paper, and the cases where the particles cloud is lit from behind or from above are considered separately.

Conclusion Although there is no doubt that this full-analytical method would give fast real-time results on today's graphics hardware, prohibitive limitations such as the infinite distance to the medium of point of view and the light source make this work barely usable for other applications than uniform dust clouds in outer space scenes. Blinn's probabilistic approach to compute the brightness of particles makes the dust rings analogous to a mere homogeneous medium.

6.1.2 Single-scattering based on shadow volumes

Several of the first works involved a homogeneous scattering medium covering the whole scene space and within which lie arbitrary surfacic objects. The two next methods that we discuss simulate beams of light by combining Frank Crow's shadow volume algorithm [Cro77] with a

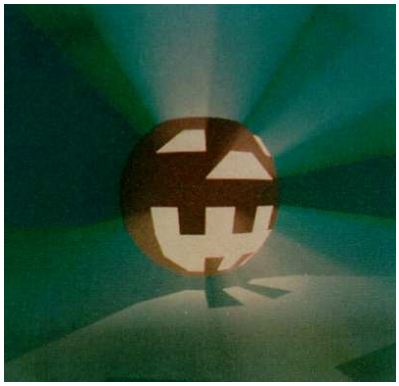

Figure 6.2: A Jack O'Lantern illuminated from inside generates light beams through a homogeneous medium. By Max [Max86].

homogeneous medium, together with a scanline rendering scheme. This was a good choice, since shadow volumes natively generate information about which portions of the space are lit or shadowed which is volumetric, indeed almost already adapted to an use with participating media. Except the next method by Max [Max86], all works discussed in this section use a ray-tracing based algorithm to calculate the final image.

6.1.2.1 Illuminating a forest using shadow volumes and radial scan lines (N. L. Max, 1986)

Overview Max [Max86] simulates both the shadows cast by opaque or translucent geometry, and the single scattering of light by the atmosphere. Rendering is based on the shadow volumes algorithm, which is modified to be more adaptated to outdoor scenes lit by the sun, defined either as a directional source, or as a point light placed above the scene. Only one light source can be placed in the scene. This is relevant if we suppose that in daytime outdoor scenes, scattering effects in the atmosphere are caused by light rays coming from the sun.

Adaptating shadow volumes to radial scan lines In an algorithm based on scan lines, such as shadow volumes, all view rays associated to a row of pixels on the image plane form a scan plane, which intersects to the image plane.

Trying to render trees using shadow volumes proves tricky precisely because each leaf generates a new shadow volume, which propagates until it reaches the ground. Shadow volumes requires shadow planes, which delimit shadow volumes, to be sorted by increasing distance from the camera, but only shadow planes which may be intersected by common scan planes must be sorted together. Therefore, when a high number of leaves at the upper half of a tree cast shadows and thus create as many shadow volumes oriented vertically, nearly horizontal scan planes at the lower half of the trees may intersect a very large number of shadow planes which, then, have to be sorted together.

The first modification on the shadow volumes algorithm consists in scanning the screen plane radially, as illustrated in figure 6.3, instead of using classical row-by-row scheme. Instead of advancing from top to bottom, scan planes are represented in polar coordinates with a radius r and an angle μ. When scanning the image plane, μ evolves counter-clockwise and scan planes

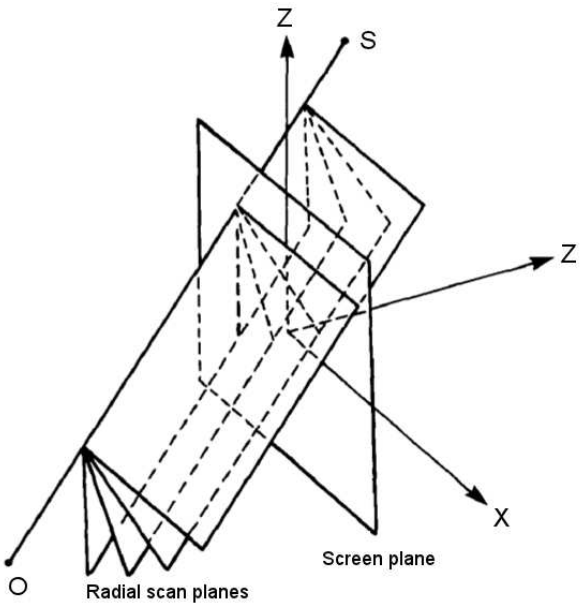

Figure 6.3: Radial scan lines around axis OS and intersecting the screen plane. Modified from fig. 1 in [Max86].

describe a rotation around axis OS between the observer's position O and the light source S.

Because radial scan planes will always be parallel to sun rays, the number of shadow planes crossed by each scan plane is considerably reduced. This change is designed to improve rendering speed by minimizing the number of elements that need to be sorted together at each frame.

Rendering atmospheric illumination In this method, atmosphere is simply modeled as a homogeneous medium (i.e. with a constant density) that covers the whole scene, which reduces the complexity of the rendering phase. However, the overall algorithm remains not trivial because of the need to sort shadow edges of scan planes and object polygons.

The atmosphere is illuminated by single scattering, which implies only one scattering deflection of light rays by the medium.

The author distinguishes two cases, when using a punctual light source, and when using a directional source, and propose two distinct models.

In case of a directional light source, all light rays arrive parallely on the scene.

When light travels through the medium on a differential distance $\mathrm{d}t$, part of the energy is scattered, depending on the scattering coefficient K, and another portion is absorbed, which is directly proportional to the medium's density ρ.

Light $\mathrm{d}L(O)$ scattered along a differential distance $\mathrm{d}t$ on the view ray is simply given by :

$$\mathrm{d}L(O) = K J_H \exp\left(-\rho(H - t\cos(\mu))\sec(\varphi)\right)\exp(-\rho t)\,\mathrm{d}t, \tag{6.1}$$

where J_H is the intensity of the light at altitude H from the observer (the source is at an infinite distance), t is a distance on the view ray from the observer, μ is the angle between the view ray and the vertical, *sec* denotes the trigonometric *secant* function, and φ is the angle between light rays and the vertical.

Therefore, the light $L(O)$ in-scattered by the medium between two positions t_i and t_{i+1} and reaching the observer is given by :

$$L(O) = \int_{t_i}^{t_{i+1}} K J_H \exp\left(-\rho H \sec(\varphi) - \rho t(1 - \cos(\mu)\sec(\varphi))\right) \tag{6.2}$$

In the case of a punctual light source, it is further assumed that the medium does not absorb light. Then, the radiance $L(O)$ in-scattered is obtained fairly easily :

$$L(O) = \int_{t_i}^{t_{i+1}} \mathrm{d}L(O) \, \frac{1}{l^2 + (t - t_0)^2} \, \mathrm{d}t, \tag{6.3}$$

where l is the distance between the point light source and its orthogonal projection on the view ray.

Conclusion The main contribution of this work is the adaptation of the shadow volumes rendering technique to outdoor scenes illuminated by a single light source, the sun, positioned above the objects, mainly trees. This is done by using radial scan lines with the source's position taken as the center of rotation, instead of the classical algorithm scanning the image horizontally. Although the shadow planes generated prior to rendering remain unchanged, because the scan process is parallel to sun rays, rendering is accelerated because the number of surfaces encountered during each scan is reduced to the minimum. In comparison, classical row-by-row scan planes would intersect the vertical polygons bordering all separate shadow volumes cast by each leaf.

6.1.2.2 Illuminating a foggy scene based on shadow and light volumes (T. Nishita et al., 1987)

Overview Another related method is the one by Nishita [NMN87], who renders shafts of light in generic scenes featuring solid geometry and based on shadow volumes. However, in comparison with Max [Max86], Nishita models different media using layers of constant density : smoke at the bottom, and the atmosphere as a second layer on top of the smoke, both separated by a surface.

In this work, the illumination comes from non-isotropic point light sources, typically for rendering spotlights effects in smoky rooms. The use of parallel light sources is discussed as well.

Shadow volumes and light volumes Portions of the scene's volume which are illuminated are characterized by combining both light volumes, defined by the nature of the source, together with shadow volumes [Cro77]. The light volume of a spotlight is defined the usual way by a cone whose origin is the position of the source itself, and whose axis is the direction of the source. Directional light sources can only be used with indoor scenes, and their light volume is defined as a prism whose base is the window from which the sunlight is supposed to enter the scene.

Light and shadow volumes are precomputed, by first computing the illumination volume, which is actually directly given by the parameters defining the source (type, position and cutoff angle). Then, a shadow volume is computed for every polyhedra (convex polygon) intersecting the illumination volume of each source.

Unlike Max [Max86] who choosen to use radial scan lines, Nishita proceeds the classical way using horizontal scan lines, which is, however, a good choice when using shadow volumes for a

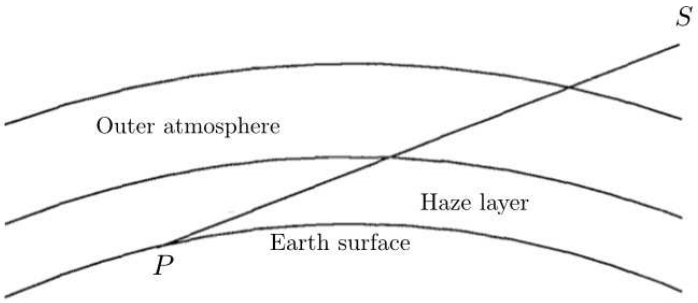

Figure 6.4: A direct light ray from the sun passes through the different laters. Modified from fig. 5 in [Kla87].

generic scene, unlike Max who optimized his method for forests lit from above. Since no multiple scattering is computed, scattered light is only evaluated for view rays which intersect a light volume. When a ray traced from a pixel intersects a light volume, the lit segments along this ray are extracted by excluding segments included inside shadow volumes from those inside light volumes.

Finally, the illuminated portions of the scattering medium are rendered simply by solving Kajiya's rendering equation [Kaj86]. The integral along the view ray only considers lit segments extracted at the previous step.

Conclusion In comparison with Max [Max86], this method allows a larger diversity of scenes and allows more artistic lighting simulations using spotlights. Both smoky indoor and foggy outdoor scenes can be rendered, but the use of homogeneous layers spreading over the whole scene horizontally does not allow modeling more compact media such as smoke or clouds.

6.1.3 Atmospheric rendering using analytical layers

We pursue with a series of works simulating the effects of sunlight due to the presence of the atmosphere around the Earth, sometimes assumed completely flat [Kla87]. The atmosphere is always modeled as one [KONN91] or two [Kla87] layers of horizontally constant density, which allows for an analytic computation of light scattering, mostly limited to single-scattering. In comparison with works presented previously, virtual scenes are not generic and do not contain solid geometry other than the Earth's surface, defined as a horizontal mesh. Instead, these methods focus on trying to obtain realistic optical results, for example by taking into account the different behaviours of sun rays caused by different wavelengths, or by simulating the decrease in density with the gain in altitude within the atmosphere.

6.1.3.1 Atmospheric lighting using a flat Earth and two homogeneous scattering layers (R.V. Klassen, 1987)

Overview Klassen [Kla87] begins with a very interesting work introducing spectral treatment in his approach to atmospheric visualization. Each pixel color is sampled at thirty-three different wavelengths, using a sampling step of ten nanometers.

Two different layers The atmosphere is divided in two parts, corresponding to two distinct concentric layers of different thickness :

- The haze-filled layer, or inner atmosphere, containing both air molecules and particles resulting from pollution or fog.
- The haze-free layer, or outer atmosphere, only containing air molecules.

Because they are considered separately, Klassen is able to apply a different scattering phase function to each layer. The upper atmosphere is mainly composed of small isotropic scattering particles such as air molecules and water droplets in suspension, for which Rayleigh scattering is used. However, the hazy lower layer is filled with larger non-spherical particles which scatter more light, and which angular scattering function is more complex because they present more than one scattering lobe. In this case, Klassen chooses to work with a precomputed table of values.

Light rays entering the atmosphere have to cross both scattering layers before reaching the observer's eye, as illustrated in figure 6.4. However, single-scattering is only simulated within the haze-filled layer, whereas the reduction of the intensity of sunlight due to out-scattering is simulated for both layers. Since Rayleigh scattering is not computed, the blue of the sky is modeled as a constant color which variations only due to scattering in the haze-filled atmosphere.

Conclusion One of the main drawbacks of Klassen's atmospheric model is its assumption that the Earth is completely flat. In the reality, because the Earth is round, no view ray can be parallel to the Earth's surface. In Klassen's model, when one looks at the horizon, instead of having the view ray exiting the atmosphere into the outer space, the observer is unrealistically able to view objects which would otherwise be occluded due to the rotundity of the Earth. Another missing aspect of this method is the illumination of the sky, even if this upper atmospheric layer attenuates sunlight. The illumination of the upper atmosphere would later be computed by most other atmospheric rendering methods, using Rayleigh's phase function of the Henyey-Greenstein approximation. Klassen's work will later be continued by Inakage [Ina89].

6.1.3.2 Single-scattering illumination of clouds and a single round analytical atmospheric layer in different conditions (K. Kaneda et al., 1991)

Overview Kaneda [KONN91] tries to fix some of Klassen's method drawbacks, and to achieve a more realistic atmospheric illumination by taking into account more phenomena. The roundness of the Earth is taken into account, and instead of using two layers of different homogeneous density, the atmosphere is modeled as a single spherical layer, with a density decreasing exponentially with altitude. Several atmospheric conditions are studied, and switching from one to the other is made by changing the parameters of the atmosphere's density function. The main phenomena rendered here are : direct sunlight diffusely reflected on objects, and single-scattering of sunlight through fog as well as clouds (modeled using a specific model). The color of sunlight changes with the position of the sun to the horizon.

Reflections on the geometry In this work, both diffuse and specular reflections on objects are computed. Specular reflections of sky light by objects are computed using the Cook-Torrance model [CT82], which has the advantage to consider the spectral distribution of radiance. The sky

Figure 6.5: Direct sunlight is scattered by fog. The presence of the buildings generates light beams. By Kaneda et al. [KONN91].

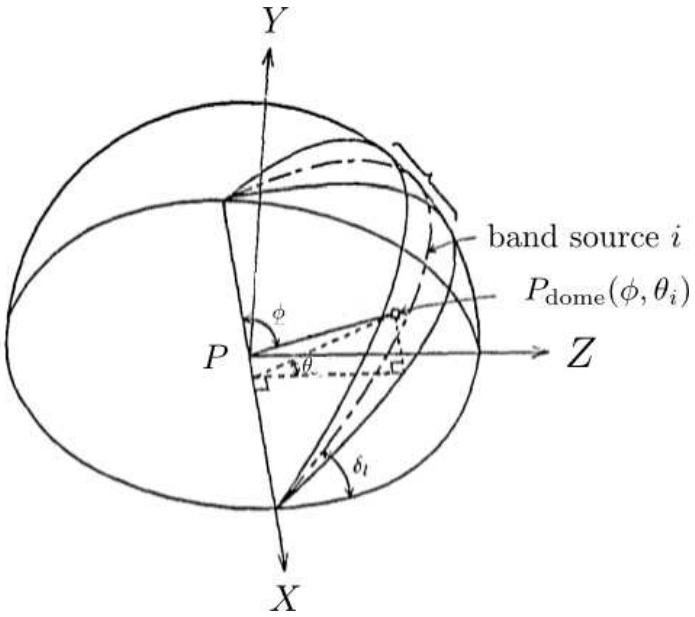

Figure 6.6: The sky dome, divided in N light emitting band sources. Modified from fig. 1 in [KONN91].

is itself treated as a hemispheric light source, called *sky dome*, and which is composed of a number of band sources with a relatively large width, like in [NN86]. The sky hemisphere is divided in N light emitting bands, which intensity does not vary along its width but only along its length.

We define ϕ and θ as the angles between the band source and respectively the x-axis and the horizontal plane (illustrated in figure 6.6). Because the radiance emitted by each band remains constant when θ varies, we instead use θ_i as the center value of θ for band source i.

$L_{\text{spec}}(P)$, the radiance incoming from the sky dome and which is reflected in the direction of the camera by the object at P, can be obtained by summing the reflection of all elementary amounts of radiance $J(P_{\text{dome}}(\phi, \theta_i))$ emitted from each differential position $P_{\text{dome}}(\phi, \theta_i)$ on the sky dome.

$$L_{\text{spec}}(P) = \sum_{i=0}^{N} \frac{1}{N} \int_0^{\pi} R_s(\phi, \theta_i, \vec{\omega}) J_{\text{sky}}(P_{\text{dome}}(\phi, \theta_i)) \sin(\phi)^2 \, \mathrm{d}\phi, \tag{6.4}$$

where $R_s(\phi, \theta, \vec{\omega})$ is the object's specular reflectance, $\vec{\omega}$ is the direction of the view ray, $J_{\text{sky}}(P_{\text{dome}}(\phi, \theta))$ denotes the radiance directly emitted by the sky dome from position $P_{\text{dome}}(\phi, \theta)$ on its hemisphere.

Because this equation presents multiple sums and integrals, Kaneda proposes a technique to reduce computation time by only integrating over a limited area of the sky dome. Actually, the major part of the light reflected specularly in the direction of the observer originates from an area of the sky dome from where the main incident ray was emitted. This area is simply constituted by all patches of the dome surface from where sky light reflects on the surface of objects above a given arbitrary threshold.

Rendering beams of light through fog In this work, beams of light through clouds are visible because of the presence of the atmosphere or fog in the scene and around objects. The fog is visualized using the classical single-scattering model [Kaj86], and the phase function is assumed uniform. However, another major difference in comparison with Klassen [Kla87] is the presence of a second hemispheric light source, in addition to the sun modeled as a parallel source. Both sources actually illuminate the medium, and their respective contributions have to be taken into account :

$$L_{\text{fog}}(O) = L_{\text{fog, sun}}(O) + L_{\text{fog, sky}}(O) \tag{6.5}$$

Concerning $L_{\text{fog, sun}}(O)$, the single-scattering of direct sunlight by the fog, because the phase angle becomes fixed due to the directionality of the sun, Kaneda is able to derive an analytic solution.

However, $L_{\text{fog, sky}}(O)$, the light emitted by the sky dome and which is scattered by the fog towards the camera can be expressed as :

$$L_{\text{fog, sky}}(O) = \int_0^{\|OP\|} J_{\text{sky}}(x(t)) \; \beta(x(t)) \exp\left[-\tau(O, P)\right] \mathrm{d}t, \tag{6.6}$$

where $\beta(x(t))$ is the local attenuation coefficient (which includes both absorption and out-scattering), and $\tau(A, B)$ is the optical depth between positions A and B.

The incoming sky light $J_{\text{sky}}(x(t))$ can be written as :

$$J_{\text{sky}}(x(t)) = \sum_{i=0}^{N} W_i \int_0^{\pi} J_{\text{sky}}(P_{\text{dome}}(\phi, \theta)) \sin(\phi) \exp\left[-\tau(x(t), P_{\text{dome}}(\phi, \theta))\right] \, \mathrm{d}\phi, \tag{6.7}$$

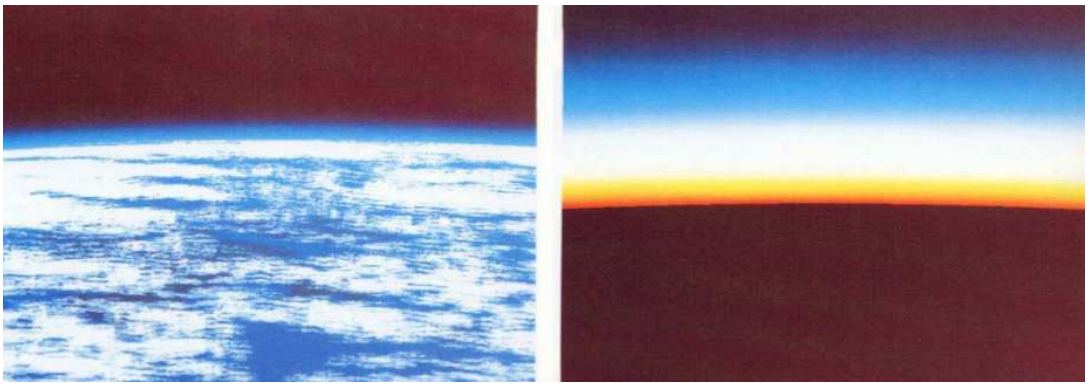

Figure 6.7: Close-ups of light scattering by the atmosphere. By Nishita et al. [NSTN93].

where W_i is the width of band source i.

Conclusion This atmospheric rendering method simulates a wide range of natural phenomena. In comparison with Klassen, participating media such as air molecules and fog are modeled using a much more physically accurate functions, decreasing exponentially with the altitude. Sparse clouds are also added in the scene, modeled by mapping a sum of sines waves on an ellipse, and specular reflections on the geometry are also rendered. However, one thing that remains unclear is the extend in which spectral sampling is taken into account, as the sampling method is not described in details.

6.1.3.3 Rendering atmospheric scattering, concentric layers of clouds and the illumination of both the Earth's surface and the oceans (Nishita et al., 1993)

Overview Atmospheric scattering continues with Nishita et al. [NSTN93] who visualize the Earth from outer space and try to simulate several major phenomena :

1. The appearance of the atmosphere.
2. The appearance of the Earth viewed from outer space.
3. The appearance of the oceans, for which this method was the first to seriously consider the sea not as just a surface, but as a scattering volume.

Each phenomenon is actually visualized using single scattering only, multiple scattering is neglected. Like in Kaneda's method [KONN91], the density of air molecules and aerosols evolves with the altitude.

Single-scattering through the atmosphere We define O_{atm} the position where the view ray penetrates the atmosphere. The radiance $L_{\text{atm}}(O)$ due to the atmosphere is calculated using the classical single-scattering equation :

$$L_{\text{atm}}(O) = \int_{O_{\text{atm}}}^{P} L_{\text{in}}(x(t)) \exp\left[-\tau(x(t)O_{\text{atm}})\right] \mathrm{d}t, \tag{6.8}$$

where $L_{\text{in}}(x(t))$ is the in-scattered radiance at position $x(t)$ on the view ray, obtained by :

$$L_{\text{in}}(x(t)) = J(S_{\text{atm}})KF(\alpha)\rho\frac{1}{\lambda^4} \exp\left[-\tau(x(t)S_{\text{atm}})\right], \tag{6.9}$$

where $J(S_{\text{atm}})$ is the direct illumination from the sun at S received at the position S_{atm} where sunlight enters the atmosphere, K is a constant parameterizing the type of atmosphere, ρ is the local density ratio depending on the altitude, λ is the wavelength of the light, $F(\alpha)$ is the phase function of the atmosphere, and α is the scattering angle.

Light reflected by the Earth The Earth is not self-emitting light, but receives both direct lighting from the sun and indirect scattered lighting from the atmosphere. The radiance reflected by the Earth's surface $L_{\text{earth}}(O)$ is calculated by :

$$L_{\text{earth}}(O) = r \ \left(\cos(\delta)J(S_{\text{atm}})\text{exp}\left[-\tau(S_{\text{atm}}P)\right] + J_{\text{sky}}(\delta)\right), \tag{6.10}$$

where r is the diffuse reflection of the Earth.

$J_{\text{sky}}(\delta)$ is the radiance transmitted from the sky to the surface after being scattered into the atmosphere, and δ is the angle of incidence between the incoming ray and the normal to the surface.

Several concentric layers or clouds are added to the color reflected by the surface, which are generated by a Mandelbrot set of fractals. The light reflected by these clouds is the sum of, on the one hand, sunlight that is in-scattered on the clouds, and light reflected by the surface that is attenuated by the clouds.

The color of the sea takes into account both the sunlight reflected at the surface of the water, and light refracted within the water which is then scattered and attenuated by water particles. Part of this light eventually gets deflected towards the camera and is refracted one more time before finally reaching the viewpoint.

Conclusion Similarly to Kaneda [KONN91], the atmosphere's density is designed so that it decreases exponentially with the altitude. An interesting aspect of Nishita's work is however the simulation of the interaction of sunlight with the water, taking into account both the reflections by the surface of the sea and the scattering of sunlight by the water as a bounded homogeneous participating medium. Clouds are also rendered, but again cannot be shaped arbitrarily by the user, as their density is modeled using fractals mapped on concentric spheres.

6.1.3.4 Accurate single-scattering of sunlight through the atmosphere based on spectral wavelength sampling and spectrum reconstruction (J. Irwin, 1996)

Overview Irwin [Irw96] goes further than Klassen [Kla87], Kaneda [KONN91] and Nishita et al. [NSTN93] with a more physically realistic modelization of both the atmosphere's density distribution and the solar radiation. Rayleigh's scattering coefficient and phase function are used to simulate the scattering of light rays by air molecules.

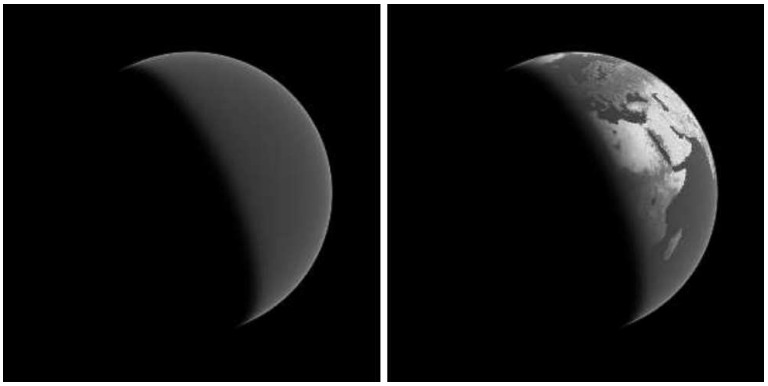

Figure 6.8: Left : Scattering of sunlight by the atmosphere alone. Right : Both the textured Earth and the atmosphere are rendered. By Irwin [Irw96].

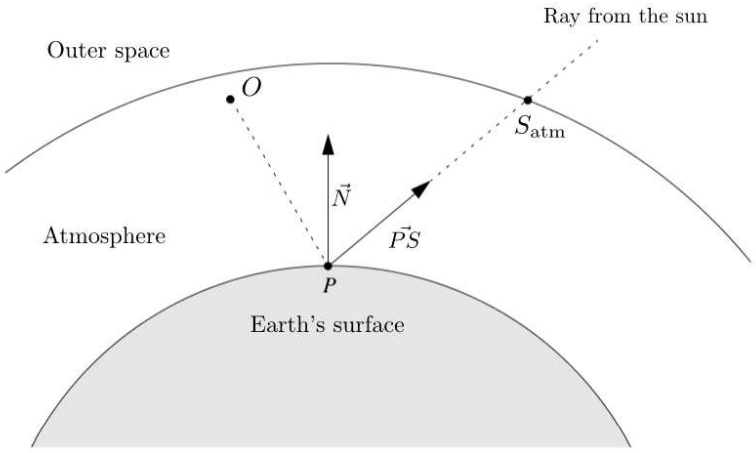

Figure 6.9: Reflection of light rays on the Earth's surface. Modified from fig. 2 in [Irw96].

As usual, the Sun is represented as a parallel light source, and the atmosphere is modeled as a single layer with a fixed height. Only single-scattering is considered, therefore indirect lighting on the Earth's surface due to sunlight scattered in the atmosphere is not computed. Moreover, absorption by the ozone layer is neglected. Shadows from the Earth on the atmosphere are rendered.

If the elements are similar to those of previous works, the challenge lies in working with spectral wavelength sampling to perform all computations, and the fact that the observer is not fixed on the surface of the Earth, but can watch the scene from outer space as well.

Radiance reflected by the Earth The Earth is represented by a simple sphere, on which is mapped a RGB texture. If we omit the appearance of the medium itself but rather focus on that of the geometry, the radiance $L_{\text{earth}}(O)$ arriving at the observer's position O and due to the reflection of incoming radiance by the earth's surface (figure 6.9) can be obtained as :

$$L_{\text{earth}}(O) = J(S_{\text{atm}})\Gamma_D(\vec{N_P} \cdot \vec{v}) \exp\left[-(\ \tau(S_{\text{atm}}, P) + \tau(P, O)\ )\right], \tag{6.11}$$

where $J(S_{\text{atm}})$ gives the direct lighting from the sun at position S_{atm} where the light first enters the atmosphere, $\vec{N_P}$ and $\vec{v}$ are respectively the normal to the surface and the incident light direction, $\tau(A, B)$ gives the optical depth between points A and B, and P is the position where the view ray intersects the Earth's surface. Here, we have already taken into account out-scattering by the medium along both the incident path from S_{atm} to P and from P to O. Γ_D is the surface's diffuse reflectivity, which corresponds directly to the RGB components in the texture.

Radiance in-scattered along the view ray Following the usual single-scattering scheme, to compute the visual contribution $L_{\text{atm}}(O)$ of the atmosphere itself, we start from the observer and integrate all radiance in-scattered along the view ray until we reach the ground or exit the atmosphere :

$$L_{\text{atm}}(O) = \frac{1}{4\pi} J(S_{\text{atm}}) \int_O^P K(r(t), \alpha) \exp\left[-(\ \tau(S_{\text{atm}}, x(t)) + \tau(x(t), O)\ )\right] \mathrm{d}t, \tag{6.12}$$

where $r(t)$ and $x(t)$ are respectively the unitary volume and the position situated at distance t from the observer on the view ray, and $K(r(t), \alpha)$ is the scattering coefficient corresponding to scattering angle α.

The scattering function K used in this work is that of Rayleigh. Please refer to the original paper [Irw96] where the author gives a detailed introduction to Rayleigh scattering.

Conclusion Apart from the self-shadowing by the Earth, this method does not render more phenomena than the three previous works that we studied. Only single-scattering illumination is considered, and also lack clouds to add details like previous methods did. However, its visual realism is brought by the very sophisticated spectral sampling of sunlight, including a reconstruction of the spectrum of the final pixel color using a natural spline interpolation, and a final conversion to RGB components using the CIE XYZ color space.

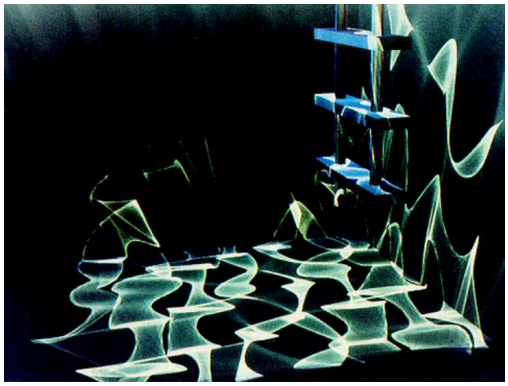

Figure 6.10: Underwater effects due to light beams generated by the refraction of light by the water surface. By Watt [Wat90].

6.1.4 Underwater lighting using a single homogeneous layer

During the years 1990s, another category of techniques related to homogeneous media illumination was the simulation of underwater lighting effects. We choose to discuss the two main noticeable works, starting with Watt [Wat90] and then Nishita and Nakamae [NN94].

The two techniques simulate the two same effects :

- Caustic reflections caused by sunlight arriving on the water surface and being refracted into the water and then illuminating underwater geometry.
- Beams of light through single-scattering homogeneous water, which is the only participating medium in this type of scene.

However, both methods visualize the result using different rendering algorithms.

6.1.4.1 Visualizing underwater light beams and caustics (M. Watt, 1990)

Overview Watt [Wat90] considers a pool and renders two main underwater lighting effects : caustics at the bottom of the water, and shafts of light as a volumetric effect within the water itself, as a scattering medium.

This was one of the firsts methods attempting to simulate caustic reflections, i.e. lighting of diffuse surfaces by transmission from specular surfaces. The two dominant approaches in illumination were, at the time, radiosity and ray-tracing. Radiosity ignores specular reflections and considers all surfaces as diffuse, and ray-tracing does not compute any reflection on diffuse surfaces, therefore neither of these two techniques can be used to render caustics. In this work, Watt uses backwards beam tracing, which is a modified version of the *backwards ray-tracing* algorithm, suggested by Arvo [Arv93].

Light beams and caustic polygons Caustic effects are caused by light being transmitted (refracted) by the water surface which is modeled as a specular animated surface.

In a first pass, which is view-independent, all caustic polygons are created. This is done by first building incident light beams, one for each water surface polygon. Incident light beams are

delimited at their top by a single point, the position of the light source, and at the bottom by their respective polygon. Rays casted from the source towards each polygon vertex define the bounding edges of the beam.

Transmitted light beams are then created by reflecting or refracting those rays based on the normal to the surface at each vertex, which generates the bounding edges. The caustic effect caused by each transmitted beam illuminates the first diffuse polygon it intersects (for example, floor at the bottom of a pool). The intersection between a transmitted beam and a diffuse polygon results in a caustic polygon, which does not form additional geometry to existing objects, but are used as additional shading information at the rendering step.

Rendering In a second pass, the scene is rendered using a standard rendering algorithm like ray-tracing, with the first modification that caustic polygons obtained at the first pass are rendered together with all other objects. Caustic effects having already been computed as polygons, a large choice of rendering methods can be used.

The caustic polygon is shaded depending of the intensity I_c it receives :

$$I_c = I_s \ \vec{N_s} \cdot \vec{L} \ \frac{A_s}{A_c,} \tag{6.13}$$

where I_s is the intensity if the light source, $\vec{N_s}$ is the normal vector at the specular surface, $\vec{L}$ is the incident direction of the light on the surface, A_s and A_c are areas of respectively the specular and the caustic polygon.

The second element that is rendered is the single scattering of light beams by the water, i.e. the effect leading to the visualization of underwater shafts of light, in other words the volume of each beam must be rendered. Beams are modeled as light volumes, like in previous works by Nishita [NMN87], and their contribution is obtained by integrating light in-scattered along each segment of the view ray that passes through a light beam.

Conclusion This underwater rendering method is designed for a particular type of effect, which can easily be encoutered in a wide range of applications, provided that the water is modeled by its surface as an animated mesh. However, the very specific rendering algorithm based on backwards ray-tracing might be difficult to integrate within a more global rendering framework.

6.1.4.2 Underwater light scattering based on the accumulation buffer (T. Nishita and E. Nakamae, 1994)

Overview Nishita and Nakamae [NN94] extend the previous work by Watt [Wat90]. Here again, only single scattering is considered. Although the water surface is discretized in a grid of triangles of identical area (two triangles subdivide each grid cell), underwater objects are modeled using metaballs (sometimes referred as blobs, introduced in [Bli82a]) built on a function of potential, with which testing intersections with rays is convenient.

Light volumes Similarly to Watt [Wat90], each water surface triangle primitive is the top boundary of a light beam, here called light volume, which edges start at each surface vertex at the top, and are directed towards the bottom in the refracted direction of a light ray coming from the sun

Figure 6.11: Light beams illuminate underwater objects modeled using metaballs. By Nishita and Nakamae [NN94].

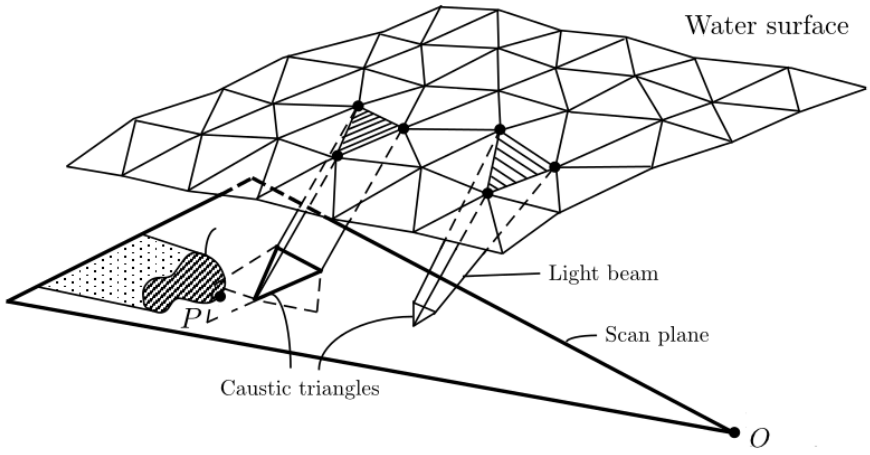

Figure 6.12: Intersection between a scan plane and light beams. Modified from fig. 2 in [NN94].

and hitting the surface at this vertex. In the absence of waves, the water surface is plane, and because of the directionnality of the sun, all light volumes are parallel to each other. When water is animated with waves, the surface grid gets deformed, and in this situation, although all light volumes stay connected so that every point in the space under the water is inside at least one light volume, refraction directions may converge or diverge, and so do light volumes.

Rendering Rendering is performed line by line, and for each of them we start by determining the intersection between the corresponding scan plane and each light volume edge (figure 6.12). The intersection between a scan plane and a light volume forms a caustic triangle, the element which scatters light towards the camera plane and makes shafts of light visible.

The whole rendering technique is clearly based on rasterization, and the use of a depth buffer to discard hidden surfaces and hidden caustic triangles. The radiance in-scattered by each triangle towards the camera is evaluated at each of its three vertices, and a linear interpolation is performed to approximate a value at each pixel covered by the projection of the triangle on the screen plane.

The authors emphasize the use of the accumulation buffer, which is a frame buffer used to sum pixel values. In this method, the accumulation buffer contains the final image which forms step by step by iteratively adding up the contribution of each caustic triangle on the pixels it covers. Actually, all scan planes may be parallel to each other, but to obtain a solution to the single scattering equation [Kaj86], all radiance quantities in-scattered along each view ray needs to be integrated, which is approximated by summing intensities of all caustic triangles belonging to each scan plane.

The radiance $L(O)$ reaching each pixel at O is obtained using the following model :

$$L(O) = L(P)\ \exp(-\beta\|OP\|) + \int_0^{\|OP\|} L(x(t))\ \exp(-\beta\ t)\mathrm{d}t, \tag{6.14}$$

where $L(P)$ and $L(x(t))$ are respectively the radiance reflected by the nearest object intersected at position P and the radiance in-scattered at position $x(t)$ towards the pixel at position O. We note β the extinction coefficient which is constant all over the medium.

In-scattered radiance $L(x(t))$ is obtained by :

$$L(x(t)) = \left[J(Q)\ T(\theta_i, \theta_t)\ \frac{A_s}{A_c}\ K(\alpha)\ \exp(-\beta\ \|Qx(t)\|)\right] + L_a, \tag{6.15}$$

where $J(Q)$ is the radiance received by the surface at position Q, $T(\theta_i, \theta_t)$ is the refraction function giving the portion of incident light transmitted from incident angle θ_i towards refracted angle θ_t, $\frac{A_s}{A_c}$ is the form factor (ratio between the specular triangle area and the caustic triangle area) and $K(\alpha)$ is the phase function of the water.

Conclusion Modeling underwater objects using metaballs is a good choice, since curved shapes are very common in natural underwater environments. Based on rasterization and the accumulation buffer, in comparison with Watt [Wat90], this other underwater rendering method may be easier to implement inside a modern rendering engine using OpenGL and shaders on the GPU without any major adaptation.

6.2 Heterogeneous media techniques

6.2.1 Introduction

6.2.1.1 About our classification

After homogeneous volumes, we now turn to techniques considering the illumination of arbitrarily user-defined heterogeneous media. In the previous section, we did already encouter analytically heterogeneous media having their density evolving according to a global function [KONN91] [NSTN93] [Irw96]. Instead, in this second section we discuss methods which allow the user to completely shape the density distribution locally, based on a sum of independant discrete elements.

We chose to separate these methods in three categories :

- Methods using a function basis, where the density is modeled as a set of particles, each particle being associated to a function, usually a Gaussian, defined on a small interval and translated at the particle's center.
- Methods using a regular grid of samples, where the density at each position inside the medium is provided as a set of final precomputed values, directly fetchable in the memory or in a texture :
  - Methods only computing single-scattering illumination, using the usual two-step process. The influence of each voxel on another is limited to attenuation of direct lighting from the source or in-scattered radiance along the view-ray.
  - Methods considering a few steps of multiple-scattering illumination, which have the particularity to compute direct exchanges between pairs of samples, thus fully exploiting the voxelized form of the density function.

Before we start, it can be interesting to mention Fourier volume rendering [Mal93], a very specific technique to accelerate the visualization of heterogeneous volumetric data using the three-dimensional Fourier transform.

6.2.1.2 Fourier volume rendering (T. Malzbender, 1993)

Fourier volume rendering [Mal93] is a method to speed-up the rendering of volumetric data, based on the Fourier projection-slice theorem. According to the Fourier projection-slice theorem, the 1D Fourier transform of the projection on a line of a 2D function corresponds to an 1D slice of the 2D Fourier transform of this function.

For volumetric data to be rendered on a screen, this data has to be projected on the screen plane. The fact is that because the Fourier projection-slice theorem can be extended from a 2D image to 3D data defined spatially, we can also write that the 2D Fourier transform of the orthogonal projection of the 3D data on a 2D plane is equivalent to a 2D slice of the 3D Fourier transform of the volumetric data.

The idea behind Fourier volume rendering is :

- The 3D Fourier transform of the volumetric data to be rendered can be calculated once and for all as a precomputation process.

- At runtime, although a direct visualization of 3D data in the native spatial domain would require performing an integration along a view ray for each pixel on the screen, using this method, a projection of the 3D data on the screen can be obtained much faster by simply evaluating the precomputed 3D Fourier transform along a 2D plane.

However, there are several drawbacks to Fourier volume rendering :

- Only an orthogonal projection can be obtained, whereas the majority of rendering applications need other types of projections.
- Hidden parts cannot be handled, since the projection itself is performed by the 3D Fourier transform, which allows no control over the boundaries of the data to be projected.

Conclusion Fourier volume rendering can obviously be interesting for some specific applications such as the real-time visualization of medical data from body scans. However, the use of the Fourier transform instead of a ray-marching to integrate the density along the line of sight makes it impossible to illuminate the medium with single or multiple-scattering, and the unavoidable orthogonal projection is absolutely not adapted to realistic rendering.

6.2.2 Heterogeneous media modeled using a function basis

Applications dealing with the animation of media such as gaseous fluids or smoke are often based on a system of particles. Each particle must be able to evolve freely and continuously in a non-discrete space domain. Indeed, the domain of allowed possible positions for particles must not be restricted to discrete domain of the centers of the cells of a three-dimensional grid. For this reason, the great majority of smoke animation techniques use a set of basis functions such as RBFs [1] to model and animate the fluid. The next method is an example of fluid animation where the medium is modeled using blobs, and then directly rendered under this form, very convenient since the intersections between the view ray and spheres are easy to compute. As we will see in section 7.3.1.2, some illumination algorithms require to sample such particles inside a grid of voxels prior to rendering.

6.2.2.1 Rendering a medium made of blobs using a non-uniform ray-marching (J. Stam and E. Fiume, 1993)

Overview We begin with Stam and Fiume [SF93] trying, on the one hand, to simulate the physical process behind turbulent wind fields and, on the other hand, to visualize a gas perturbated by these wind fields. They basically animate wind fields by combining a large scale behaviour generated using a deterministic process, with small scale behaviours based on stochatic perturbations. In this method, the participating medium is the gas, which is modeled as a set of blobs and illuminated by simulating single scattering of light.

> **Note :** Since the physics behind numerical fluid simulation is not our primary focus, for more details, please see the original paper [SF93].

[1] For Radial Basis Functions, also called *metaballs* in these early works. RBFs can be based on different one-dimentional functions, the most common being the Gaussian.

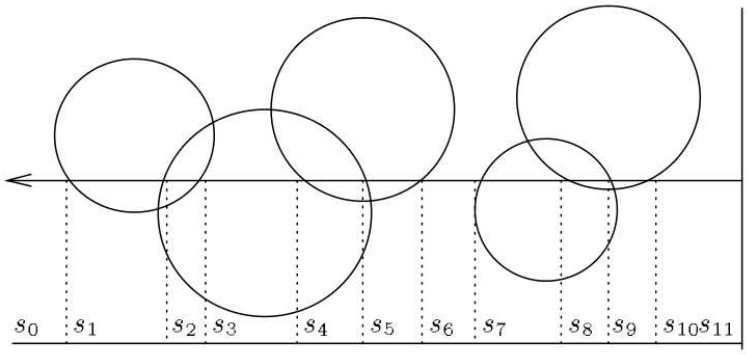

Figure 6.13: Ordering the intersections by the view ray with a series of blobs. Modified from fig. 2 in [KVH84].

Rendering The scene and the medium are visualized using a modified ray-tracing, which can render both the geometry and the gas using a front-to-back process.

To compute the visual effect of the presence of blobs between the camera and the geometry, all blobs intersected by each ray are detected, and for each of them both the entry and the exit positions are stored in a list, represented by their respective parameter on the ray (figure 6.13). The list is then ordered by increasing distance from the eye, resulting in a series of distinct intervals. All positions belonging to each interval along the ray are at the intersection of the same list of blobs.

Then, following the usual two-step scheme of single scattering [KVH84], the radiance reaching each pixel is obtained by summing all quantities of light $L_{\text{outgoing}}(s_i, s_{i+1})$ on each interval $[s_i, s_{i+1}]$, each being attenuated by the transparency factor $\tau(s_0, s_{i+1})$ associated to this interval :

$$L(s_0) = \sum_{i=0}^{N-2} \tau(s_0, s_{i+1})\ L_{\text{outgoing}}(s_i, s_{i+1}) \tag{6.16}$$

$$L_{\text{outgoing}}(s_i, s_{i+1}) = [1 - \tau(s_i, s_{i+1})]\,[\chi\ J(s_i, s_{i+1}) + (1 - \chi)L(s_i, s_{i+1})] \tag{6.17}$$

χ is the albedo of the gas, $L_{\text{outgoing}}(s_i, s_{i+1})$ is the radiance leaving interval $[s_i, s_{i+1}]$ towards the camera, $J(s_i, s_{i+1})$ is the total radiance received due to direct lighting from all sources and $L(s_i, s_{i+1})$ accounts for self-emission as well as indirect lighting.

The calculation of intersections between the view ray and the blobs is accelerated using a hierarchy of bounding spheres.

Conclusion This method is a good example of how to take advantage of the representation of the medium as blobs to accelerate the ray-marching. Typical single-scattering illumination is used, but instead of advancing regularly along the view ray, Stam and Fiume are able to use non-uniform steps such that the ray-marching jumps directly to a position situated inside a different blob intersection configuration. Unfortunately, doing so requires to construct and order a new list of intersection intervals boundaries for each pixel and for each frame, it is therefore not obvious that a brute-force ray-marching would not prove faster on the GPU.

6.2.2.2 Representing participating media in Computer Graphics (L. Da Dalto et al., 1995)

We can also mention a work by L. Da Dalto et al. [DDJC95], who did consider a new way of representing participating media. Their goal is to avoid discretizing the medium in a set of voxels, which leads to substantial errors in the distribution of the medium's density and its other properties, since their values are constant all over the volume of each single grid cell. Instead, they propose to represent participating media using a sum of ellipsoids. Exact values are specified by the user only at a reference position at the center of each ellipsoid, and both a variation direction and a variation function are specified for each property : absorption, scattering and self-emission. This results in the ability, at rendering, to obtain an exact continuous mathematical evaluation of the medium's properties at each position inside the participating medium. They consider multiple-scattering at the rendering step, using a technique similar to progressive radiosity [CCWG88], where the exchanges of energy between the sources and the ellipsoids are computed only for their central point.

Conclusion The use of progressive radiosity still makes it difficult today with this method to reach interactive time. Where this representation of the particle's properties as the combination of an exact value at the center together with a variation function is interesting is that it may allow to speed-up the rendering process by only considering the value of the center point above a given scattering order.

6.2.3 Single-scattering illumination of regularly sampled media

We now continue with media directly provided to the application as a regular three-dimensional grid of voxels. The methods discussed here do not animate the medium, but only deal with its visualization. This next work by Kajiya and Von Herzen is well-known, and was the very first to consider the illumination of heterogeneous participating media and first introduced the classical two-step process to compute single-scattering illumination. The second work in this category by Ebert and Parent is also remarkable, as the first attempt to truely combine both solid geometry and sampled volumes within the same rendering process.

6.2.3.1 Extending the ray-tracing algorithm to volumes (J. T. Kajiya and B. P. Von Herzen, 1984)

In 1984, Kajiya and Von Herzen [KVH84] are the firsts to illuminate a full-heterogeneous medium provided as a regular grid of density samples, in one of the most cited works in the domain of volume rendering. In another paper published two years later [Kaj86], Kajiya introduces his *scattering equation*, which still serves today as a theoretical basis for the illumination of scattering media in numerical simulations. This theory was later reformulated and explained by other researchers such as Arvo [Arv93], in articles that, by the way, inspired us for writing part I chapter 3 in this manuscript. It is therefore not necessary to describe it again in details here.

At the time, the ray-tracing algorithm had just been introduced in 1980 by Whitted [Whi80], allowing a realistic illumination of solid objects, by reproducing direct illumination, perfect mirror reflections and hard shadows. In the meantime, Blinn had inaugurated the research on participating media illumination in computer graphics, with his work on the simulation of the illumination

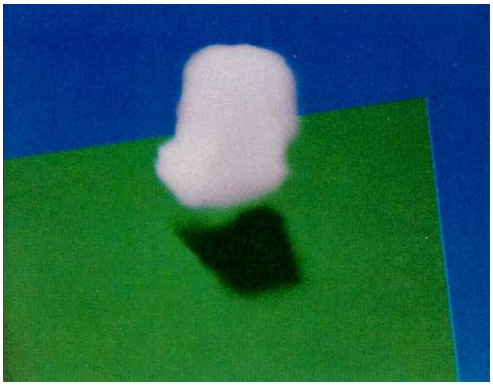

Figure 6.14: Single-scattering illumination of a cumulus cloud. By Kajiya and Von Herzen [KVH84].

of the rings of Saturn, using a very simplified analytical model, extended by other researchers [BGMR83, Vos83].

In this work, Kajiya and Von Herzen extend the classical surfacic ray-tracing to also handle volumetric data. In comparison with Blinn's model, this one was the first true grid-based volume renderer, which can be used for virtually any type of heterogeneous scattering medium, even if the kind of medium which will later be most represented as a regular grid is clouds.

They implement both a ray-tracing version of Blinn's single-scattering model, and then also consider an extension to multiple-scattering, which corresponds to a medium having a high albedo.

Finally, they present a quick algorithm to simulate the formation of clouds, in order to automatically model such media as grids of densities. It is very interesting to compare this algorithm with recent cloud simulations [DKY+00, LHCL05], which we discuss in the next chapter.

Conclusion Although thirty years later, every aspect of the method presented in this paper have now been surpassed [KPH+03], the main principle used to illuminate and render the volume remains exactly the same.

6.2.3.2 Unified rendering of both surfaces and volumes (D. S. Ebert and R. E. Parent, 1990)

Overview Ebert and Parent [EP90] extend Kajiya and Von Herzen [KVH84] and develop a method able to deal with both surface-based solid geometry, as well as volume-based objects such as gas. Previous techniques [Duf85] had already tried to handle both types of elements in the same scene, but that had only been achieved through basic compositing, where two sub-scenes were actually rendered in distinct passes.

In this method, based on scanline rendering, surfaces and volumes are illuminated and visualized together within a common process, that goes beyond simple compositing. In particular, both volumes and surfaces can cast shadows on each other.

To achieve this result, Ebert and Parent insert into the scanline algorithm a combination of both the use of an accumulation buffer and Kajiya's volume rendering equations [Kaj86]. Volumes are illuminated using single-scattering, as usual, and are treated using a different illumination and

density accumulation model depending on whether it represents a gas or something solid.

> **Note :** In this work, the medium is actually defined using procedural Perlin noise, but the entire method is perfectly adapted *as is* to the use of a grid of samples without any modification.

Volumes in the A-buffer algorithm This method is based on the scanline algorithm, where the image is rendered line by line and pixel by pixel. Solid geometry is rendered following Carpenter's A-buffer technique [Car84], which allows for anti-aliasing by taking into account the percentage of the area of the pixel that each surface fragment covers, so that more than one surface can contribute to its final color. After all fragments were ordered by their distance from the camera, the final color of the pixel is computed by summing their respective contribution starting from the farthest to the nearest.

Surface and volume elements are combined in the A-buffer by creating new fragments from the volume, and inserting them in each pixel's fragment list, between fragments generated from solid objects. After both the view ray and surface fragments have been mapped into the world space, the part of the ray covered by the medium is determined. Then, in order to solve the depth ambiguity between surfaces and the volume, the whole volume is cut in several parts along the view ray, each two parts being separated by a surface fragment, so that no surface fragment lie inside any contiguous section of the volume. Each volume section is then changed into a new fragment inserted in the list between the two surrounding surface fragments. Only visible surface and volume fragments are illuminated and rendered.

Rendering the medium This method handles solid and gaseous volumes differently. Solid volumes are illuminated in a way which is similar to the illumination of a surface, by applying an illumination model locally based on the light's position, the material of the solid volume, and its normal vector, which is determined based on the local variation of opacity. This is made possible by assuming that the medium's opacity is at its maximum inside the object it represents, and decreases as we approach its surface towards emerging from the object.

Only single-scattering of light is simulated to render gaseous media, using Kajiya's classical rendering equations [Kaj86]. The radiance $L(O)$ reaching the camera at O is discretely approximated by :

$$L(O) \simeq \sum_{t_{\text{near}}}^{t_{\text{far}}} \exp\left[-\tau \sum_{t_{\text{near}}}^{t} \rho(x(t'))\mathrm{d}t'\right] L(x(t))\ \rho(x(t))\ \mathrm{d}t, \tag{6.18}$$

where t_{near} and t_{far} are the distance from the camera of respectively the nearest and the farthest plane, τ is the optical depth of the medium, and $\rho(x(t))$ is the density at position $x(t)$ along the view ray.

The radiance $L(x(t))$ in-scattered at position $x(t)$ is approximated using :

$$L(x(t)) \simeq \sum_i J_i(x(t))\ F(\alpha_i), \tag{6.19}$$

where $J_i(x(t))$ is the direct lighting incoming from source i along the ray at $x(t)$, $F(\alpha)$ is the phase function and α_i is the phase angle corresponding to light rays incoming from source i.

The authors suggest that in order to speed up the calculations, the optical depth between each volume element of the scene and each light source can be precomputed in a table. Actually, even if the medium is animated which means that those precomputations have to be repeated at each frame, this can still accelerate the rendering of each single frame during which the same optical depth value may be required several times.

Conclusion This method is interesting since it has still kept all its relevance today. The problematic of illuminating transparent solid geometry is similar is similar to illuminating a medium. While illuminating a medium usually requires a ray-marching along the view ray from the yon[2] plane towards the eye, transparent solid surfaces intersecting the view ray must be illuminated from the farthest to the nearest. Indeed, even if we consider using modern GPU functionnalities, the whole rendering algorithm would still have to be implemented as described above since there is no built-in feature in the pipeline which would natively help rendering transparent surfaces without having to first order them.

6.2.4 Towards multiple-scattering in sampled media

In this last category of historical methods, we also discuss the illumination of media whose density function is defined as a volumetric grid of samples. The two next methods differ from those of the previous category by considering multiple-scattering, and performing direct transfers of energy between voxels. We first discuss another milestone work, the zonal method by Rushmeier and Torrance, which extends the classical radiosity algorithm by Goral so that it also handles volumes beside surfaces. We then finish this chapter with a method by Nishita et al. to illuminate clouds using both single-scattering and an approximation of multiple-scattering.

6.2.4.1 Extending the radiosity method to volumes (H. E. Rushmeier and K. E. Torrance, 1987)

Overview The zonal method [RT87] by Rushmeier and Torrance cannot be avoided. The original radiosity algorithm by Goral [GTG84] defined a new framework to simulate the transfer of light radiance between surfaces. On the other hand, the zonal method is originally the name of a technique developed by Hottel and Sarofim in 1967 [HS67] to model the transfer of heat radiations. However, the name is most often used to designate this well-work by Rushmeier and Torrance, who extend the classical radiosity to not only handle surfaces, but participating media as well.

It is extended to all three possible types of interactions : radiance transfers between surfaces, radiance transfers between volumes and interactions between a surface and a volume in both ways. Other improvements are brought to the radiosity method, such as the possibility to use non perfectly diffuse surfaces, like reflecting surfaces and directional light sources.

New geometric factors The original radiosity method used form factors to express radiance transfers between surfaces. They are now extended to volumes, and are renamed *geometric factors* (figure 6.16). The authors also propose a technique to speed-up the evaluation of these geometric

[2]The *near* and *far* planes are sometimes also called respectively *hither* and *yon*.

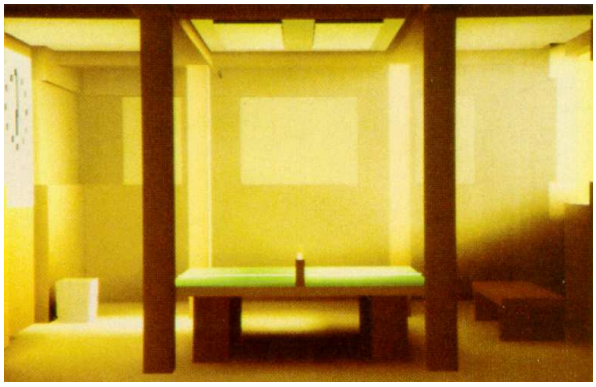

Figure 6.15: Illumination of an indoor scene using the zonal method. All types of interactions between surfaces and volumes are computed. By Rushmeier and Torrance [RT87].

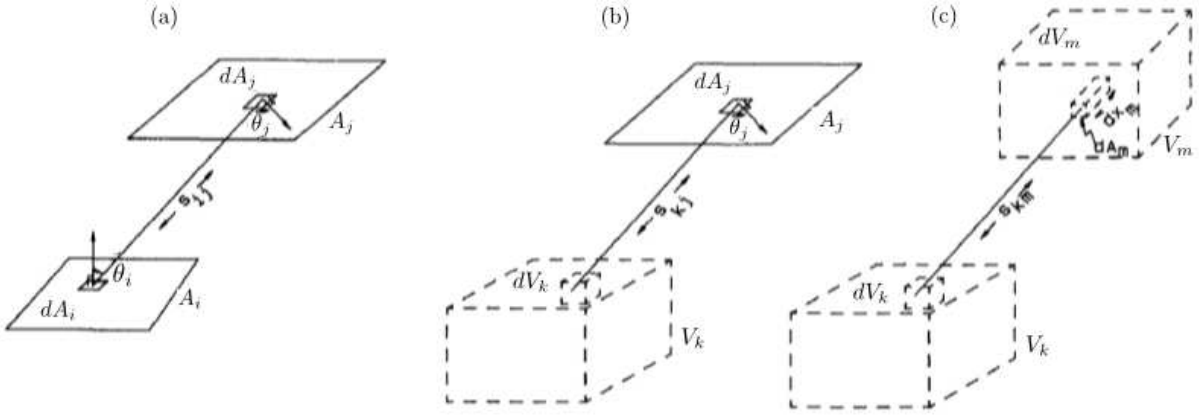

Figure 6.16: Different types of radiance transfers. (a) Interaction between two surfaces A_i and A_j. (b) Integration between a surface A_j and a volume V_k. (c) Integration between two volumes V_k and V_m. Modified from fig. 3 in [RT87].

Figure 6.17: A cumulonimbus cloud modeled using 258 metaballs is illuminated by sunset light. By Nishita et al. [NDN96].

factors based on the hemicube algorithm [CG85], which we will not describe here. While approximating geometric factors using the hemicube projection, a depth buffer can also be used to discard hidden surfaces.

Conclusion This work by Rushmeier and Torrance is inspired by the original zonal method by Hottel and Sarofim [HS67] and extends the radiosity to handle both surfaces and volumes in an unified fashion. This method can be classified as deterministic since on the one hand, no stochastic process is involved and on the other hand, the illumination of a medium implies considering every differential volume and surface element separately. Because the medium is discretized into small volumes, just like surfaces are discretized into small areas, this technique is compatible with heterogeneous media, but suffers the same computational complexity drawbacks as Goral's radiosity [GTG84]. Indeed, even if hidden surfaces are discarded, the principle itself still implies computing radiance transfers between each existing pair of mutually visible elements. When volumes are introduced, the complexity becomes even higher with the need to integrate the medium's density between each two elements. This methods is able to accurately compute all types of radiance transfers within a scene containing both surfaces and volumes, but its prohibitive complexity makes real-time still far from reachable using today's mainstream GPU's.

For this reason, we will not discuss further extensions of the radiosity.

6.2.4.2 Single and multiple-scattering illumination of clouds modeled using metaballs (T. Nishita et al., 1996)

Overview Nishita et al. [NDN96] model clouds as metaballs and illuminate them by simulating sky light and multiple scattering of sunlight. Clouds are also illuminated by light reflected by the ground, and the computation of overall radiance reaching each pixel is accelerated using a stochastic process. Clouds are treated as inhomogeneous participating media, and several phase functions are considered, which are chosen depending on the size of the scattering particles.

Model used for single-scattering in clouds We can start by considering the model used for single-scattering of light by cloud particles, also denominated 1^{st} order scattering. It simply

corresponds to the classical single-scattering model, which is used at each bounce of the light over a particle in case of multiple scattering. The radiance $L(M_a)$ due to single scattering of sunlight in the cloud and received at position M_a where the view ray exits the cloud towards the eye is given by :

$$\begin{aligned} L(M_a) &= L(M_b)\ e^{-\tau(M_b,M_a)} \\ &\quad + \int_{t_a}^{t_b} J(x(t))\beta\rho(x(t))F(\alpha)\ e^{-\tau(\rho(x(t)),M_a)}\ \mathrm{d}t, \end{aligned} \tag{6.20}$$

where M_b is the intersection between the view ray and the cloud boundary which is the farthest from the eye, $\tau(A, B)$ is the optical depth between positions A and B, $x(t)$ is the position associated to parameter t along the view ray, t_a is the parameter associated to position M_a, t_b is the parameter associated to position M_b, $J(x(t))$ is the direct lighting coming from the sun at position $x(t)$, β is the attenuation coefficient, $F(\alpha)$ is the scattering phase function, and α is the phase angle.

The light $J(x(t))$ incoming at each position along the ray is given by :

$$J(x(t)) = J(M_c)\ e^{-\tau(M_c,x(t))}, \tag{6.21}$$

where M_c is the position where the incoming sunlight first crosses the cloud's upper boundary.

The Henyey-Greenstein phase function In this work, several phase functions are used, due to the varying size of particles. Rayleigh scattering is used for atmospheric molecules, and Mie scattering for scattering by aerosols and pollution. Instead of using two distinct models, Nishita et al. use a common phase function, the Henyey-Greenstein function, which makes it possible to switch from one type of scattering to the other by simply changing a physical parameter g in the expression. They use an improvement by Cornette [CS92] of this model :

$$F(\alpha, g) = \frac{3(1-g^2)}{2(2+g^2)} \frac{1+\cos(\alpha)^2}{(1+g^2-2g\cos(\alpha))^{3/2}} \tag{6.22}$$

Whether Rayleigh or Mie scattering is used, both phase functions lead to a strong forward scattering.

Multiple scattering using voxels and Monte Carlo sampling The most interesting feature of this illumination algorithm by Nishita et al. is the combination of a discrete space and stochastic sampling of light paths.

This space containing the medium is first subdivided in a grid of voxels, and the idea of the algorithm is to compute exchanges of energy between these discrete volume elements, using a modification of Rushmeier's volume-to-volume form factor [RT87], adapted to anisotropic scattering by taking into account the phase function.

Instead of computing all exchanges between all voxels in the medium, a sample space or exchange pattern is precomputed, which is then used to speed-up the computation of energy exchanges between voxels in the whole cloud. Typically, this exchange pattern is created by considering one of the voxels in the pattern as the reference voxel, and precomputing some of the interactions of other voxels with this reference voxel, considered as a receiver. We first determine

which voxels may directly scatter significant energy towards the reference voxel, and which voxels may only send an amount of radiance which is small enough for this voxel to be neglected. For each voxel affecting the reference voxel, the form factor between this voxel and the reference voxel is computed and stored in the pattern. Contribution ratios of light arriving at the reference point from each voxel by 2^{nd} and 3^{rd} order scattering paths are also computed and stored in the pattern.

The optical depth between the sun and each voxel in the cloud space in computed at runtime, as well as the optical depth between each pair of voxels. Then the pattern is placed on each voxel to determine the radiance scattered towards the eye. Finally, the total radiance scattered towards the eye is obtained by summing all radiance quantities in-scattered along the view ray, using the classical single-scattering model (equation 6.20).

To further speed-up the whole process, only a portion of all non-negligible light exchanges between pairs of voxels is actually computed. These pairs of voxels are randomly selected using Monte-Carlo sampling among all path with a high contribution ratio, and the result of the integration of in-scattered radiance is slightly corrected using a compensating factor.

Conclusion This method is a good example of the combination of both a function basis and a grid of samples. Because they are particularly adaptated to the round shape of clouds, Nishita at al. first model the medium using a set of metaballs. If we consider the method by Stam and Fiume [SF93], we see that single-scattering can be computed directly on particles. Multiple-scattering, however, implies intermediary steps of radiance exchanges between volume elements which need to be smaller than the metaballs used to shape the medium. Furthermore, storing and accessing data in a grid of voxels is made easy by the direct relationship between a position in the scene and the corresponding address in memory.

6.3 Conclusion

Analytical media techniques An important part of volume illumination methods developed before the appearance of modern programmable graphics hardware dealt with homogeneous or analytically defined media. As we have seen, for an analytical definition to be possible, the number of parameters in the model is strongly reduced in comparison with the original equation of transfer, as so is its genericity and adaptability. Indeed, all methods discussed in this section feature participating media which are only adaptated to the particular scene and situation (atmospheric laters, a pool of water, etc.).

Because of the simplicity of the models, there is little doubt that these methods would run in real or interactive time using modern GPUs. However, because they cover the whole scene and the analytical definition does not allow for local adjustments in the medium's density, these methods do not appear flexible enough to be exploitable for our problematic : outdoor fog, animated smoke, etc.

Heterogeneous media techniques In comparison, heterogeneous media are generally not compelled to a particular type of application. Methods based on particles such as [NDN96] can be highly parameterized (type of basis function, size, amplitude, etc.) and can therefore be used to design a large variety of phenomena. Some articles [KVH84, EP90] do not even mention a specific type of medium which the presented method is intented for.

Although a regular grid of samples is straightforward to evaluate and can be stored in a variety of structures such as volumetric textures, it requires much more memory space since even empty zones inside the medium are modeled, generally with a density equal to zero. Instead, a list of particles whom only the position of the center together with a few properties are stored is much more optimized, since a large low-frequency volume can be designed with only a handful of particles.

In the next chapter, to have a clearer idea of the advantages and drawbacks of these representations, we review modern real-time illumination techniques, and switch our focus on the different strategies to reach real-time using programmable graphics hardware.

Chapter 7

From 2000 : Towards real-time volume rendering

> "It is not Justice the servant of men, but accident, hazard, Fortune–the ally of patient Time–that holds an even and scrupulous balance."
>
> Joseph Conrad, *Lord Jim*

In this chapter, we address more recent techniques published since the beginning of years 2000s, when modern GPUs appeared and researchers started to use built-in features from the rendering pipeline to speed-up and parallelize computations. Whereas in the previous chapter we focused on the different possible representations for a participating medium that could be encountered in the early literature, we chose to separate these recent works according to the type of medium considered.

We identified three categories of media most related to our work :

- **Fog**, and large scale outdoor phenomena. They can be modeled analytically and some cover the whole space without having clear mathematical boundaries. Their density distribution varies only very softly across the scene, and never approaches full opacity locally.

- **Clouds**, and related outdoor atmospheric media. These volumes are well delimited spatially, relatively dense and possess a high albedo. Most of the time, they are only illuminated by a single directional light source, representing the sun.

- **Smoke**, and smaller animated media. This kind of medium is characterized by high-frequency details, and a rapidly changing aspect. It is generally encountered in indoor scenes and can be illuminated from within by one or more point light sources.

While we dedicate a section to each type of medium, we discuss the different techniques employed to achieve interactivity or real-time, thanks to native capabilities of GPUs as well as the programmable functionnalities of the pipeline.

7.1 Real-time rendering of large-scale participating media

We begin our review of GPU-based real-time participating media rendering methods with two types of large phenomena : outdoor fog and outer-space dust. Although the two phenomena are different in nature and origin, they have some aspect in common, that they can both be heterogeneous [BMA02, MHLH05] as well as modeled analytically [LMAK00, Bli82b], and most of the time remain static.

7.1.1 Real-time rendering of outdoor fog

Historically, fog has already been used in real-time simulations for a long time, actually since general public three-dimensional video games appeared in the early 1990s, to reduce the visibility distance and thus making it possible to do not draw objects far from the observer. In this case, because the fog was to go almost unnoticed by the player and also be as cheap as possible in terms of performances, a homogeneous medium was used, with a color matching the environment of the scene to make its presence plausible and simulate environment lighting. Such a simple fog can be computed in the image space in a post-process, is very easy to implement today using programmable shaders, and is even already present as a native feature on hardware pipelines such as OpenGL, including the culling of the geometry above a fixed distance from the eye.

We review four very different methods. In [LMAK00], the fog has a constant density, and the model is reformulated in angular terms to allow for precomputations to be able to speed-up the illumination. The rendering is then further accelerated by only computing the illumination at sampled points and using hardware texture mapping to interpolate non-computed values. In [BMA02], although not illuminated, the fog is fully heterogeneous and modeled in a function basis, before being rendered using hardware texturing and blending. In [Zdr04], the fog is designed by combining several octaves of Perlin noise, precomputed in 3D textures. A Cg couple of shaders is used to visualize the result in real-time. Finally, we briefly see in [ZCS07] why using the depth buffer to retrieve the camera's position in the scene is actually inaccurate, and overview the solution that they propose.

7.1.1.1 Real-time single-scattering homogeneous fog based on an angular reformulation of the base model (P. Lecocq et al., 2000)

Overview We begin fog rendering with Lecocq et al. [LMAK00] who simulate real-time single-scattering illumination of mobile point light sources and directional sources in a homogeneous medium, and propose an application for testing headlights in a driving simulator. Real-time is achived by reformulating Nishita's model [NMN87] in angular terms, which makes it possible to approximate part of the equation with a polynomial which coefficients can then be precomputed, and vary according to the type of fog. The reformulated model then only depends on a few parameters which can be obtained very easily. Rendering is finally achieved thanks to the simple mapping of a volumetric texture.

Nishita's model We start with the following simple model, from Nishita [NMN87] :

$$L(O) = e^{-\beta l} L(P) + \int_0^l e^{-\beta t} \beta L_{\text{in}}(x(t)) \mathrm{d}t, \tag{7.1}$$

Figure 7.1: The homogeneous single-scattering fog creates halos around street lights. By Lecocq et al. [LMAK00].

where $L(O)$ is the radiance received by the observer at position O from view ray direction $\vec{\omega}$, β is the extinction coefficient of the homogeneous medium, l is the distance from the pixel to the nearest object in front of the camera, $L(P)$ is the radiance reflected by the object at position P, t is the parameter associated to the position x on the view ray, and $L_{\text{in}}(x)$ is the in-scattered radiance at position x, which is obtained by :

$$L_{\text{in}}(x) = \frac{\chi}{4\pi} \sum_{i=1}^{N} e^{-\beta r_i(t)} \frac{I_i(\vec{v})}{r_i(t)^2} F(\alpha_i(t)), \tag{7.2}$$

where χ is the albedo of the medium, $r_i(t)$ is the distance from x to source i, $I_i(\vec{v})$ is the intensity of source i emitted towards incident direction $\vec{v}$, and $F(\alpha_i(t))$ denotes the phase function of the medium.

Reformulation : from incrementing parameter t to incrementing incident angle θ First, Lecocq et al. reformulate Nishita's equation so that the integration variable is changed from a parameter t to an angle θ. After this reformulation, the new integral does the same job, and still involves advancing step by step on the view ray from the surface to the camera to evaluate and sum local in-scattered radiance. However, every position on the view ray is now defined by the angle between the direction perpendicular to the view ray and the vector from the source to this position. Following this scheme, θ_d is the angle associated to the camera, θ_0 is the angle asociated to the object, θ is the angle associated to the moving position on the ray.

We now have (see [LMAK00] for details) :

$$L(O) = e^{-\beta l} L(P) + \beta \frac{\chi}{4\pi} \frac{e^{-\beta m}}{h} \int_{\theta_d}^{\theta_0} e^{-\beta h \frac{\sin(\theta)+1}{\cos(\theta)}} I(\theta + \nu) F_{\text{angular}}(\theta + \frac{\pi}{2}) \mathrm{d}\theta, \tag{7.3}$$

where m is the parameter corresponding to the projection on the ray of the position of the source, h is the distance from the source to its projection on the ray, ν is the angle associated to an arbitrarily chosen reference direction $\vec{\omega_{\text{ref}}}$, and $F_{\text{angular}}(\theta + \frac{\pi}{2})$ is the new phase function adaptated to the new integration variable θ, and I is the intensity of the unique light source in the scene.

Polynomial approximation From the reformulated equation, we now consider the term situated inside the integral, on which we perform a limited development to the third order.

We obtain :

$$L(O) \approx e^{-\beta l} L(P) + \beta \frac{\chi}{4\pi} \frac{e^{-\beta m}}{h} \left[c_0(\beta, h) \int_{\theta_d}^{\theta_0} I(\theta + \nu) \mathrm{d}\theta + c_1(\beta, h) \int_{\theta_d}^{\theta_0} I(\theta + \nu) \mathrm{d}\theta + c_2(\beta, h) \int_{\theta_d}^{\theta_0} I(\theta + \nu) \mathrm{d}\theta \right] \tag{7.4}$$

From the full third order polynomial (see [LMAK00] for details), three constants can be identified, c_0, c_1 and c_2. Approximated in such a form, the type of participating medium can easily be modified by affecting the corresponding values to these constants that can be precomputed. Lecocq et al. provide the values for three types of fog.

Rendering At the rendering step, the volumetric illumination is computed in a 3D texture at sparsely sampled points on the screen. This volumetric texture is then applied on the geometry, and hardware texture functions naturally interpolates between neighbouring samples to map the illumination on all the geometry of the scene.

Conclusion A homogeneous fog corresponds to the model already implemented on hardware pipelines such as OpenGL, and since it covers the whole scene, it cannot be exploited to design other media than outdoor fog. However, contrarily to other real-time methods to come [BMA02, Zdr04, ZCS07] where the result is a simple blend between the medium's color and the color of the surfaces, Lecocq et al. compute the single-scattering illumination of the medium, and also consider several types of sources. Thanks to the angular reformulation, this simple single-scattering homogeneous fog becomes noticeably optimized considering the features available on GPUs at this time.

7.1.1.2 Real-time non-illuminated heterogeneous fog modeled using a function basis (V. Biri et al.)

Overview We continue with the work by Biri et al. [BMA02] on heterogeneous fog modeled with a polynomial function basis. No interaction with light sources is considered, and the scattering of light by the medium is simply globally approximated by the color of the fog.

Complete fog model They start with this single-scattering equation, where the radiance $L(O)$ received by the camera situated at position O is given by :

$$L(O) = L(P)\ \tau(O, P) + \int_0^{\|OP\|} J(S)\ \rho(x(t))\ \tau(O, x(t))\ \mathrm{d}t, \tag{7.5}$$

where P is the position of the nearest surface in front of the camera, $\tau(A, B)$ is the optical depth between positions A and B, $J(S)$ denotes the direct lighting from the sun situated at position S, $\rho(x(t))$ is the extinction coefficient at a position $x(t)$ on the view ray.

Figure 7.2: Real-time heterogeneous mist modeled using a basis of cosine functions. By Biri et al. [BMA02].

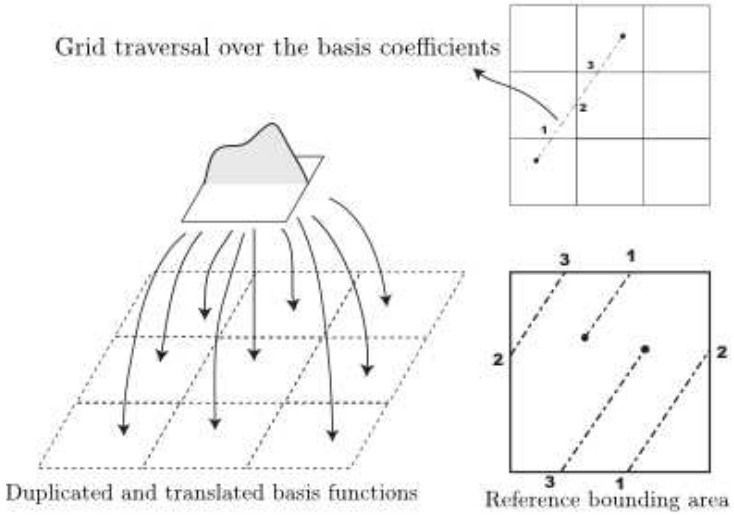

Figure 7.3: Left : a pattern function is multiplied with each coefficient to evaluate the local density. Top-right : at the rendering step, a ray-marching is performed on the grid. Bottom-right : at each ray-marching step, the density is integrated analytically between the two intersections with the cell's border. Modified from fig. 3 in [BMA02].

Simplifications to reach real-time To reach real-time, some simplifications on the model cannot be avoided. For instance, in-scattering of sunlight is not computed, but rather approximated by a constant radiance quantity L_{fog}.

Biri et al. actually perform calculations at the rendering stage using the classical fog model :

$$L(O) = L(P)\ \tau(O,P) + L_{\text{fog}}(1 - \tau(O,P)) \tag{7.6}$$

The optical depth $\tau(P,O)$ is expressed as :

$$\tau(O,P) = \exp\left[-\int_0^{\|OP\|} \rho(x(t))\ \mathrm{d}t\right] \tag{7.7}$$

Modeling the fog Designing the fog means defining its extinction function $\rho(x(t))$, which is directly linked to the distribution its density. Biri et al. study the possibility to model $\rho(x(t))$ using several types of function bases (figure 7.3). In this case, evaluating $\rho(x(t))$ requires a sum over all basis functions :

$$\rho(x(t)) = \sum_{i=0}^{N} c_i f_i(x(t)) \tag{7.8}$$

Their goal is, in order to speed-up the rendering process as mush as possible, to be able to obtain the optical depth between the viewer and the nearest surface analytically, i.e. without having to perform a numerical integration. They identify three types of function that meet this requirement :

- Cosine functions, which are naturally periodic, have a wave-like shape and are easily integrable analytically.
- Polynomial functions, which, although not periodic naturally, are duplicated all over the fog's area to simulate some periodicity. The drawback with polynomial functions is a higher computational cost.
- Equipotential functions, which can be used to design a fog which shape corresponds to a particular object, such as a point, line or sphere.

Rendering and animating the fog To insert the fog in the scene, the color of the solid geometry of the scene must be blended with the color of the fog. To achieve this, four main steps are needed :

1. The solid geometry is rendered, and the depth buffer is retrieved, which will indicate the maximum distance beyond which the fog must not be rendered because it is occluded by the geometry.
2. The coordinates of the observer and the point aimed by the camera on the geometry are determined in the global scene space.
3. The color of the fog in front of each pixel is computed analytically, and a texture is created with the contribution of the fog in the scene from the current point of view.

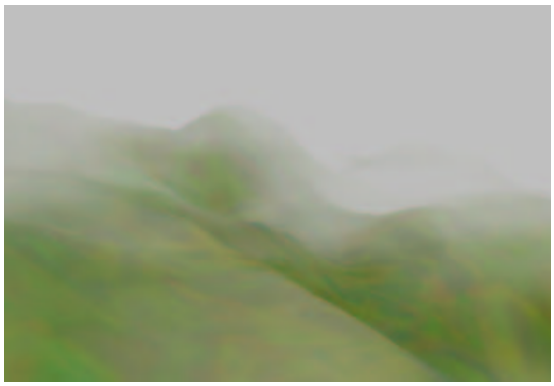

Figure 7.4: Real-time heterogeneous fog modeled using several octaves of three-dimensional Perlin noise, and rendered using Cg shaders. By Zdrojewska [Zdr04].

4. A plane is drawn in front of the camera, on which the fog texture is displayed. Hardware blending is used to mix the fog's color with the geometry behind the fog.

Because the fog's extinction function is modeled using a function basis, animation is now very convenient since after defining a few parameters such as a wind speed and a direction, interesting effects can be achieved by simply sliding the basis functions in the scene space.

Conclusion Although the fog is not illuminated, this method is interesting in the perspective of modeling light outdoor mist. Using a grid of density coefficients allows an incomparable variety of medium shapes, but to avoid the square voxels not being too visible a high resolution is necessary, which can quickly become costly in terms of memory. Rather than simply blurring the density by averaging neighbouring coefficients like in [LMAK00], using the grid values as coefficients in a function basis allows recovering a continuous distribution from a discrete set of coefficients.

7.1.1.3 Real-time non-illuminated heterogeneous fog using 3D Perlin noise and shaders (D. Zdrojewska, 2004)

Overview Instead of OpenGLSL or HLSL, Cg was the language of choice of the first publications involving programmable GPUs. One of the very firsts is the one by Zdrojewska [Zdr04], who models and renders in real-time heterogeneous fog using Perlin noise [Per85] and graphics hardware.

The fog's model No illumination or scattering of light is computed, the fog model instead only performs a simple blend between the color of the geometry and the albedo of the fog.

The color of the fog is obtained as the combination of the heterogeneous density $g_{\mathrm{p}}(P)$ at the nearest surface at position P and the linear attenuation using a homogeneous density factor g_{c} :

$$L_{\mathrm{fog}}(O) = \exp\left[-\|OP\|\ g_c\ g_p(P),\right] \tag{7.9}$$

where $L_{\mathrm{fog}}(O)$ is the color of the fog as seen by the observer, O is the position of the camera, and P is the position of the nearest surface.

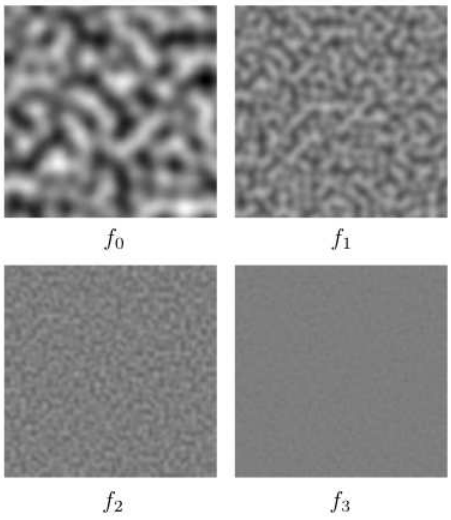

Figure 7.5: Perlin noise octaves. Modified from fig. 3 in [Zdr04].

Modeling fog using Perlin noise The density function $f(x)$ of the fog is modeled in three dimensions. Noise created using Perlin's algorithm has a high frequency, which is unnatural for outdoor since it always has a rather smooth appearance. Instead of performing a simple linear interpolation between random samples, which is not sufficient, another method proposed by Perlin [Per85] can be used. This method consists in summing together N noise functions $f_i(x)$ as layers of details, each function $f_{i+1}(x)$ having a noise frequency which is twice that of the previous layer $f_i(x)$. For this property, the N Perlin noise functions are called *octaves* (figure 7.5).

$$f(x) = \sum_{i=0}^{N-1} \frac{f_i(x)}{2^i} \tag{7.10}$$

The amplitude of each layer $f_i(x)$ is divided by 2^i to reduce it to half that of the previous layer $f_{i-1}(x)$.

Rendering algorithm Zdrojewska's rendering algorithm is rather simple. First, as a precomputation on the CPU, the N octave functions are created as N 3D textures, which is the common way of transmitting precomputed values to the GPU. The rest of the process involved is performed using a vertex / pixel shader pair.

Then, in the vertex shader, the position in the scene of both the observer and the nearest surface are computed, as well as the texture coordinates corresponding to each Perlin noise texture. The distance between the observer and the geometry is also computed at this point.

In the pixel shader, the final color of each fragment is computed by first summing the values in the textures at the surface's position to obtain the heterogeneous component and replacing the result in the equation of $L_{\text{fog}}(O)$, and then by blending the color of the fog with the color of the solid geometry.

The author also implemented a simple animation of the fog, by translating the layers of details regularly but each one at a different speed, which simulates the effect of a more complex animation.

Conclusion This method provides a good basis for anyone interested in developing a simple fog synthesis algorithm using GPU shaders. Its nevertheless suffers several drawbacks, the most noticeable being the absence of integration of the density along the view ray, which could be achieved easily provided the position in the scene of both the viewer and the surface fragment, which the authors already compute. Instead, only the coefficient present locally at the object's position is taken into account, and the attenuation due to the density between the surface and the viewer is only estimated analytically from the distance of the surface from the camera, like with a homogeneous medium. However, when the density is modeled using random high-frequency noise, because the sum of the values traversed by a ray through the grid would tend to a function of the length of that ray, such a method may be relevant for the high-frequency layers of details. But this quick analytical approximation can provide inaccurate visual results with regard to the coarser low-frequency noise layers.

7.1.1.4 Real-time non-illuminated analytical fog using shaders (T. Zhou et al., 2007)

Zhou et al. [ZCS07] try to improve basic fog rendering algorithms such as OpenGL's native fog model, who's main problem lies in the use of the values stored in the depth buffer to obtain the depth of each fragment.

Indeed, because the screen plane is a rectangular plane and cannot be reduced to a point such as the human eye, errors in the depth estimation arise when the camera is slightly rotated. When the screen plane itself is rotated at its center, pixels on the rotation axis keep their position in the scene unchanged, while others near the border of the screen are actually displaced significantly. This displacement can make a simple rotation of the screen lead to undesired changes in the depth of some fragments.

To correct these effects, Zhou proposes, instead of using the depth buffer to obtain the depth of each fragment, to compute the actual distance between the camera and each fragment. To do this, un-projecting both positions by computing an inverted projection matrix would provide an exact result, but the authors propose a less computationally expensive approach to achieve the same result, by directly extracting useful parts from the inverse projection matrix (refer to [ZCS07] for more details). The OpenGL camera properties gives information about the depth of the two *near* and *far* clipping planes, which is transmitted to a fragment shader used to display the fog, and where it is easy to obtain the actual coordinates in the scene of both the fragment and the camera, and then be able to compute their Euclidian distance. Three basic fog models covering the whole scene are treated : homogeneous fog, layered fog, as well as heterogeneous fog modeled by means of a simple product of three independent density functions, one associated to each axis.

Conclusion The idea described in this work has the advantage of being exploitable in many screen-space post-processing methods using shaders, provided that they use the standard projection matrix. However, the discussed fog models (homogeneous, layer-based and the product of independant axis-bound functions) do not actually allow designing true two-dimensional or three-dimensional fog.

Figure 7.6: Illumination of the dust around a star as the main light source. By Magnor et al. [MHLH05].

7.1.2 Interstellar objects rendering

Real-time multiple-scattering illumination of static interstellar dust (M. A. Magnor et al., 2005)

Overview Magnor et al. [MHLH05] develop a method to visualize interstellar dust forming nebulae around stars in interactive time. The medium is modeled in a grid of volume elements, and a hierarchical pyramid is generated from the original set of voxels, where the medium is represented at multiple resolutions. This multi-resolution pyramid then forms the basis for an approximation of anisotropic multiple-scattering. Wavelength dependance and trajectory-dependent extinction are also considered.

The point of view is not fixed, as the observer can fly through the illuminated nebula. However, an important drawback of this method lies in the precomputation of the local illumination of the medium, since this aspect imposes to keep the distribution of the medium fixed. Moreover, in this work, apart from the medium, no solid geometry is present in the scene, therefore this method can only be used to visualize fixed participating media in outer-space.

Illumination model for single-scattering The illumination model used by Magnor et al. to render each layer of the multi-resolution pyramid simply corresponds to the two-step single-scattering illumination process. Multiple-scattering is approximated discretely as a post-processing step in the image space, on the basis of the results of single-scattering illumination, therefore the model below only considers single-scattering and is not sufficient to obtain the final result.

The radiance $J(x(t))$ directly reaching each voxel at position $x(t)$ evolving along the view ray is obtained by :

$$J(x(t)) = \frac{\Phi_{\text{star}}}{4\|x(t)S\|^2}\tau(x(t), S), \tag{7.11}$$

where Φ_{star} is the total radiance emitted in all directions by the source situated at S.

The optical depth $\tau(x(t), S)$ between each voxel along the view ray and the source is expressed as :

$$\tau(x(t), S) = \exp\left[-\int_0^{\|x(t)S\|} \rho(x'(t'))\mathrm{d}t'\right], \tag{7.12}$$

where $\rho(x'(t'))$ is the medium's density at $x'(t')$, a position evolving along the direct lighting ray from each voxel to the source.

The radiance $L(O)$ resulting from single-scattering and reaching the observer at O is given by :

$$L(O) = \int_0^l \tau(O, x(t))J(x(t))F(\alpha)\mathrm{d}t, \tag{7.13}$$

where l is the maximum viewing distance from the observer, $F(\alpha)$ is the phase function, and α is a scattering angle.

Rendering algorithm with multiple-scattering approximation During rendering, multiple-scattering is approximated locally using Monte-Carlo simulation to estimate the scattering probability distribution $P(\rho, \alpha)$.

At the preprocessing step, a pyramide of multiple resolutions of the voxels grid defining the medium is generated, where each voxel contains a scattering depth quantity corresponding to the voxel's size in the scene. To generate each level n, the level $n-1$ is down-sampled so that the resolution of level n is half that of level $n-1$ along all three dimensions, and each value affected to the voxels of level n is an average of its corresponding $2^3 = 8$ voxels from level $n-1$, which is also ultimately multiplied by two to account for the difference of voxel size between the two levels.

The nebula is first rendered using single-scattering illumination only. To accelerate this process, per-voxel direct illumination is also precomputed for each level of the multiple-resolution pyramid, which corresponds to the first step of the classical two-step single-scattering illumination algorithm.

Then, all remaining computations are performed in the image space :

1. From the pyramid of directly illuminated volumes $V_0, ..., V_n$ is generated a pyramid of images $I_0, ..., I_n$, each image I_i being the result of the visualization of volume V_i, illuminated using single-scattering.

2. Difference images $\Delta I_0, ..., \Delta I_n$ are generated by substracting to each image I_i a down-sampled version $\bar{\mathrm{I}}_i$ of the image I_{i-1} from the upper level. These images $\Delta I_0, ..., \Delta I_n$ only contain the radiance scattered from seven neighbouring voxels at level $i-1$ due to the average before down-sampling.

3. A second series of difference images $\Delta\hat{\mathrm{I}}_0, ..., \Delta\hat{\mathrm{I}}_n$ is obtained by up-sampling difference images $\Delta I_0, ..., \Delta I_n$, so that $\Delta\hat{\mathrm{I}}_0, ..., \Delta\hat{\mathrm{I}}_n$ represent the radiance scattered from seven neighbouring voxels at the same level i. At this point, we have for each level, on the one side, single-scattering illumination results $I_0, ..., I_n$, and on the other side, images $\Delta\hat{\mathrm{I}}_0, ..., \Delta\hat{\mathrm{I}}_n$ containing additional illumination resulting from n very approximated neighbour-to-neighbour multiple-scattering step between voxels.

4. The final result is obtained by summing up the full-resolution image I_0 from single-scattering illumination, together with the additional radiance from all multiple-scattering levels $\Delta\hat{\mathrm{I}}_0, ..., \Delta\hat{\mathrm{I}}_n$, by a successive sum and result up-sampling process until reaching level 0.

Figure 7.7: Single-scattering illumination of a volumetric object using half-angle slicing. By Kniss et al. [KPH+03].

Conclusion This is one of the milestone methods in the field of real-time discrete volumetric data rendering using the two-step single-scattering illumination scheme on the GPU. If we consider our own work which will be presented later in this manuscript, the big disadvantage of Magnor's technique is the fact that the medium has to remain fixed and cannot be animated. One of the major problems in the illumination of a participating media, and which is certainly the most expensive component of the algorithm when real-time is the target, is the computation of the direct illumination at each voxel in the medium. This problematic is simply solved here through precomputations. Nevertheless, Magnor achieve more at runtime than simply performing a ray-marching along the view-ray to finalize the evaluation of the single-scattering equation, and actually try to approximate local multiple-scattering through a sophisticated combination of different resolutions of the precomputed direct illumination.

7.2 Real-time rendering of clouds and bounded volumes

In this section, we review several techniques for illuminating and rendering clouds in real-time. All methods use graphics hardware, most do not employ programmed shaders, except [Bou08] due to the specificity of its model.

We begin with *half-angle slicing* [KPH+03], a very popular technique, and a very interesting manner of accelerating the illumination and rendering of small sampled volumes lit from outside, which is then used by other cloud rendering works [REK+04]. In a second time, we see a simple method for automatically and inexpensively reproducing the cloud formation cycle in real-time, without involving complex fluid simulation. Finally, we discuss further specific techniques for pure visualization of clouds, designed with very different models.

7.2.1 A direct volume rendering method

Half-angle slicing and multiple-scattering (J. Kniss et al., 2003)

Overview A well-know modern bounded volume illumination technique is the one by Kniss et al. [KPH+03]. Single-scattering is faithfully approximated using half-angle slicing [KKH02], and

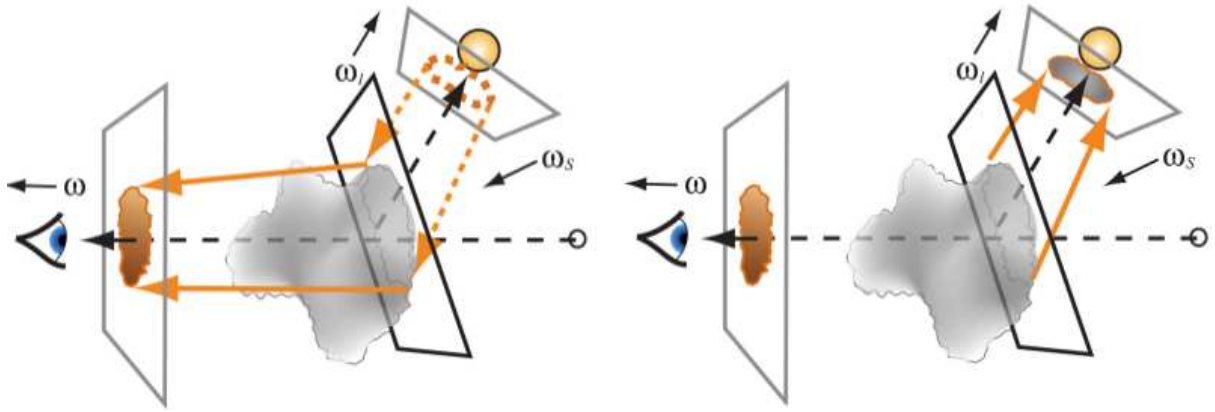

Figure 7.8: Single-scattering using half-angle slicing (ω : view direction, ω_l light direction, ω_s slicing direction). Modified from fig. 3 in [KPH+03].

multiple-scattering is approximated empirically as a straightforward extension to that same technique. Several major volumetric effects are rendered, such as chromatic attenuation and translucency. Furthermore, the authors also present a method to add details to low-resolution volumes.

When rendering surfacic geometry, direct lighting is computed using the BRDF of the material. When rendering volumetric geometry, a phase function is used instead of a BRDF, therefore by analogy to non-solid media, visualizing direct lighting of a volume involves single-scattering of light. In this work, the phase function is precomputed in a lookup table, and stored as a texture.

Single-scattering using half-angle slicing In this method, Kniss et al. first propose an algorithm to compute single scattering illumination of a small-scale medium, called *half-angle slicing*, as an alternative to the classical two-step numerical integration of radiance in-scattered towards the eye. The main principle behind half-angle slicing is to approximate the direct illumination of the volume and the accumulation of in-scattered radiance along the view ray in an unique step. The whole process is performed by progressively cutting the volume into slices oriented halfway between the view direction and the incident light's direction, starting from the slice closest to the source and advancing step-by-step towards the slice closest to the camera.

Each iteration illuminates and renders one slice :

1. The slice is rendered from the light's point of view (figure 7.8 left), and each sample on the slice is illuminated with the direct incident radiance, while the volumetric object's material properties are taken into account. This first step is necessary in order to associate to each sample on the slice the attenuation by all previously processed slices, which was accumulated on the source's framebuffer.

2. The slice now illuminated with incident radiance is rendered from the eye's point of view (figure 7.8 left), and the local in-scattered radiance is accumulated on the eye's framebuffer.

3. The light's framebuffer is updated by taking into account the additional attenuation by the slice (figure 7.8 right).

The key idea behind this technique is to avoid integrating several times over the same direct lighting rays by re-using the previous results as the main rendering loop passes from one pixel row to the other. In comparison with the standard single-scattering algorithm, because renderings from the light's point of view are involved, the volume must be fully visible by the source, and therefore distant from it. However, this is perfectly acceptable for many applications.

Extending half-angle slicing to approximate multiple-scattering Higher scattering orders are approximated by a global indirect lighting term, computed by extending the half-angle slicing algorithm. While the process advances from slice to slice, additional scattering of light is allowed, which can only occur in the same direction as the slicing algorithm. To simulate chromatic attenuation, Kniss allows two different attenuation coefficients as separate medium properties for the direct and indirect lighting steps, the latter being chromatic, i.e. it possesses separate values for each RGB color component.

To approximate both single and multiple-scattering within the same process, this modified half-angle slicing requires one framebuffer for the eye, and two framebuffers for the light's point of view. One buffer is called the *current* buffer, and is only accessed for reading attenuation values, while the other, called the *next* buffer, is only used for writing RGB color components and attenuation values as seen from the light's point of view.

Single-scattering is computed exactly in the same manner as with the standard half-angle slicing, with the difference that the attenuation between the source and each slice is only stored in the alpha component of both light buffers. Storing colors in the RGB components is not needed for single-scattering, and is reserved for the approximation of multiple-scattering, by applying an iterative blur effect on the color as seen from the light source.

The modified half-angle slicing processes as follows :

1. In a first step, the contribution of the current slice is computed and accumulated. To do this, the slice to render is drawn from the eye's point of view. The direct attenuation, associated to single-scattering, is read from the light's current buffer's alpha component, and the chromatic indirect attenuation, used for multiple-scattering, is read from its RGB channels. The radiance in-scattered towards the observer is computed in the pixel shader, and blended into the eye's framebuffer according to the rendered slice's attenuation property.

2. In a second step, the next iteration is prepared by updating both attenuation values in the light's next buffer, from which the shader will read at the next iteration's first step. The direct attenuation in the next buffer's alpha channel is modified by simply summing the attenuation of the slice with the attenuation from the current buffer's alpha channel. The chromatic indirect attenuation from the current buffer is read and slightly blurred (Kniss suggests a four-pixel neighbourhood), and the result is used as incident radiance to evaluate the transfer function at the current slice, which will also account for the chromatic attenuation of the slice. This chromatic attenuation is finally stored in the color channels of the next buffer, and the two buffers are swapped to get ready to process the next slice.

Conclusion Half-angle slicing is absolutely not a bad strategy, if we consider the applications that the authors mention in the paper, such as medical volumetric data visualization, for which both the observer and the light source are always situated outside the volume. However, because

Figure 7.9: A cloud is rendered at various successive steps of its formation using the cellular automaton, from left to right. By Dobashi et al. [DKY+00].

of the need to render the scene from the light's point of view, half-angle slicing cannot simulate effects like glows around light sources within a scattering volume. Indeed, the screen plane placed at the light's point of view is not geometrically equivalent to a point source, and furthermore, half-angle slicing can barely handle illumination in all directions.

7.2.2 Animation and rendering of clouds based on a cellular automaton

The great majority of cloud rendering methods only address the rendering aspect, due to the fact that the natural cloud formation process takes a relatively long time, and that except physical simulations, static or translating clouds can already well suffice to bring a lot of realism to common applications like video games or flight simulators. Nevertheless, it could be very interesting if it could exist an algorithm that is able to approximate the physical phenomena involved in the formation of clouds without heavily impacting real-time performances.

In this section, we consider three works which are all related to each other. We start with [DKY+00], where was first introduced the idea of using a cellular automaton to generate clouds. The cloud synthesis algorithm is then improved in [LHCL05], where new rules are added to the automaton to enable new clouds to automatically appear after others have disappeared, and the illumination algorithm is also optimized through precomputations. We finish with [Dom06], where the costly metaball-based smoothing of the automaton binary states is replaced by the use of hardware texture mapping and blending.

7.2.2.1 Real-time generation and single-scattering illumination of clouds (Y. Dobashi et al., 2000)

Overview Using an automaton is a powerful strategy for fast generation of clouds. Dobashi et al. are the firsts to propose [DNO98] this principle, before they improve [DKY+00] their method by adding real-time rendering using the GPU. They are able to render shadows by the clouds, as well as visual effects such as shafts of light inside them. Clouds are modeled as voxels inside a three-dimensional grid. Since this method is based on the one by Nagel [NR92], instead of classically modeling the clouds using floating-point values to indicate the local density or extinction coefficient, each cell contains boolean flags indicating the local progression in the cloud formation cycle.

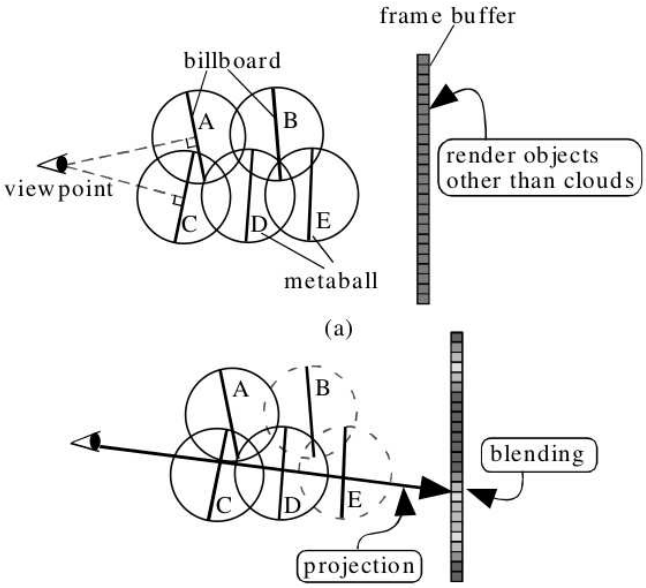

Figure 7.10: At rendering, metaballs are rendered as billboards facing the camera (top), and their density is accumulated on the frame buffer (bottom). From [DKY+00].

Animating the clouds The cloud is animated on the principle of a cellular automaton, where pre-defined transition rules are applied on the volumetric grid and modify the state of each cell. Each cell contains three boolean flags (humidity, action and cloud). Zones in the grid which may contribute to the formation of clouds have their humidity flag set to 1, and one cell in the center of the cloud has its action flag set to 1. The action flag activation process propagates from neighbour to neighbour, and each cell which has its action flag set to 1 at step t_i is ready to turn into a cloud voxel at step t_{i+1}. When a cell passes from one state to another, the previous flag is set back to 0.

After a certain number of steps depending on the local extinction probability, cloud cells undergo an extinction. To allow clouds to form again on extinct cells, the humidity and action flags are set to 1 on random cells using local humidity and action probabilities. A wind effect is simulated by simply translating cell flags within the grid.

Rendering process A continuous floating-point density distribution is first computed by smoothing the binary voxels grid, by transforming it into a function basis, where each binary coefficient multiplies a metaball.

Clouds are illuminated by computing single-scattering, using the usual two-step process (figure 7.10). In a first step, the optical depth between the sun and the center of each metaball is computed by placing the viewpoint at the sun and drawing all metaballs as billboards, textured according to their own local density function. The billboards are rendered from the nearest to the farthest from the sun, and when each billboard is drawn, because each pixel has accumulated the optical density of all previous billboards using hardware blending, its associated optical depth is given by the values in the current frame buffer. In a second step, the camera is placed at the viewpoint, the billboards are rendered in order of decreasing distance from the observer, and the radiance reaching each pixel is accumulated using hardware blending functionalities. Indeed, for each billboard, the

accumulated radiance is attenuated by its local density, before the in-scattered radiance is added to the pixel value.

This process is summarized in algorithm 1, where $J(S)$ is the radiance emitted by the sun situated at S, and $T(A, B)$ denotes the transmittance between points A and B.

Conclusion This idea of using a cellular automaton to animate the clouds is interesting for its simplicity. The visual results may not be as realistic as with a true fluids simulation, but it nevertheless manage to reproduce the global mechanism involved in the cloud formation cycle. In this work, the clouds are rendered using the classical single-scattering illumination, and the need to smoothen the density values using metaballs is certainly a great prejudice in terms of performance, although this technique by drawing and accumulating textured billboards in front of the screen plane using native blending and texture mapping functionalities of the GPU may actually prove faster today than doing the same job using loops and texture accesses in CUDA or GLSL.

Algorithm 1 Rendering metaballs with billboard splatting [DKY+00]

// Step 1 : Calculate light map texture
Setup camera at the sun, activate parallel projection
Initialize framebuffer to 1.0
Sort metaballs by increasing distance from the sun
for all metaballs c_j **do**
  Bind precomputed texture best matching the metaball's density
  Draw the metaball's billboard facing the sun
  Multiply framebuffer value with texture
  $T(c_j, S)$ = read pixel corresponding to the metaball's center c_j
  Output as 2nd target the metaball's color as $J(S) \times T(c_j, S)$
end for
// Step 2 : Rendering from the observer
Setup camera at the observer, resume perspective projection
Render solid geometry
Sort metaballs with respect to the observer's position
for all metaballs c_j **do**
  Bind metaball's color texture
  Draw metaball's billboard oriented toward the observer
  Read optical depth $T(c_j, S)$ from light map
  Blend pixel color with billboard's color attenuated with $T(c_j, S)$
end for

7.2.2.2 Improved cloud simulation based on a cellular automaton (H. S. Liao et al., 2004)

Overview Liao et al. [LHCL05] improve and speed-up the above technique by Dobashi et al. [DKY+00], mainly by adding several precomputations.

Precomputations to accelerate the rendering To speed-up the computation of the optical depth between the sun and each metaball, a Shadow Relation Table (SRT) is precomputed, which stores a series of linked lists of voxels. Each list in the SRT is associated to a ray shot from the point of view of the sun towards the metaballs, and indicates which voxels are traversed by this ray as the distance from the sun increases.

To also accelerate the computation of the radiance in-scattered on each metaball towards the observer's point of view, a Metaball Lighting Texture Database (MLTDB) is also created. This table contains precomputed in-scattered radiances for several sampled metaball densities and several sampled phase angles.

Changes in the animation rules Several changes are brought to the rules of the cellular automaton used by previous methods to model the formation of clouds. Instead of permanently disappearing, when a cloud becomes extinct, cells return to the initial vapor state, so that a new cloud may appear automatically without having to continually add new vapor. Only cells situated on the boundaries of the cloud can change the value of their *cloud* flag, so that cells inside a cloud cannot become extinct and create holes at its center. The number of cloud and vapor voxels is bounded.

Rendering The rendering algorithm is almost unchanged, except the use of the SRT and the MLTDB when computing the radiance received respectively at each metaball and by the camera. Prior to rendering, the density and illumination distributions are stored in an octree for faster access, instead of the regular grid of voxels.

Conclusion The introduction of the precomputed SRT in the algorithm necessitates each voxel of the medium to only be intersected by one direct light ray, which imposes the use of a directional source, illuminating the medium from outside. Because this table is precomputed, this method is not adaptated for atmospheric scattering simulation where the sun is moving around the Earth, but remains very interesting for other applications such as video games. Liao et al. do not use programmable graphics hardware, which would have provided further interesting possibilities.

7.2.2.3 Real-time illumination and rendering of clouds using "flat" 3D textures (D. Domański, 2006)

To finish with cloud animation based on cellular automata, we can mention Domański [Dom06]. The medium's density distribution is extracted from the grid of voxels and stored in textures using an object-oriented slicing algorithm, where the slices are not oriented parallely to the screen plane, but parallely to all three x, y and z axis. These textures are used for illuminating and rendering the clouds. Indeed, three different sets of slices are constructed separately, and the set which is oriented the most perpendicularly to the rays is selected. All slices are actually arranged like tiles on a single 2D texture to create a "flat" 3D texture.

In Dobashi [DKY+00], a high number of metaballs are drawn as billboard in front of the camera and their density and illumination accumulated on the frame buffer. In comparison, Domański simply has to render slices instead of metaballs. As the author explains, using slices instead of metaballs reduces considerably the number of elements composing the medium and therefore also reduces the number of blending operations needed to render the clouds.

Figure 7.11: Light beams caused by spotlights. By Dobashi et al. [DYN02].

Conclusion Mainly built on the basis of the method by Dobashi [DKY+00], this work, although the rendering algorithm simply consists in an object-aligned slicing, highlights a strategy which is not encountered very often in the litterature : spreading heavy computations over several successive frames.

7.2.3 Specific cloud models

We continue with four different real-time cloud rendering techniques, which do not address the problematic of its generation nor its animation. In [DYN02], clouds are viewed from outer-space and are illuminated and visualized based on a simple mapping of satellite data on concentric spheres around the Earth, as part of a more comprehensive atmospheric rendering framework. In [REK+04], a great variety of physically realistic types of clouds are rendered, together with most of the optical phenomena caused by Mie scattering. We finish with [Bou08], where an innovative approach to cloud modeling is presented, based on the association of a homogeneous core and a heterogeneous envelope built from a hypertexture.

7.2.3.1 Interactive atmospheric scattering and cloud visualization from outer space (Y. Dobashi et al., 2002)

Overview We return to atmospheric rendering with Dabashi et al. [DYN02], who propose a method involving graphics hardware. They consider various types of light sources including point lights, spotlights and the sun, as well as several types of scenes, i.e. the Earth viewed from outer space or simple indoor scenes like smoky rooms. They manage to render the change of color in the sky light that takes place between daytime and sunset, and add clouds to improve the realism of the visualization of the Earth from outer space.

Two different methods are used to render volumes depending on whether the observer is situated on the ground or whether the Earth is viewed from space. Wherever the camera is placed in the scene, this method employs a two-step process to render scattering volumes.

Note : We only mention the case where the camera is on the ground, please refer to the original paper [DYN02] for full details about all cases.

Observer on the ground : model When the observer is on the ground, light being scattered through the atmosphere creates light beams. Different effects are considered when the Earth is viewed from space. In this case, radiance $L_{\text{gnd}}(O)$ due to light beams and received at the observer's position O is given by :

$$L_{\text{gnd}}(O) = L(P)g(O,P) + L_a + L_s(O), \tag{7.14}$$

where P denotes the intersection between the view ray and the nearest surface, $g(O, x(t))$ is a function giving the ratio of light in-scattered at position $x(t)$ on the ray with respect to the total radiance emitted from the source in this direction, and L_a is the ambiant lighting in the scene.

The total radiance $L_s(O)$ in-scattered along the view ray towards the camera is obtained with :

$$L_s(O) = \int_0^{\|OP\|} J(S)H(x(t))g(O,x(t))\mathrm{d}t, \tag{7.15}$$

where $J(S)$ stands for the direct radiance emitted from the source at S towards the view ray, and $H(x(t))$ is a boolean visibility function.

The function $g(O, x(t))$ is expressed as :

$$g(O,x(t)) = \rho F(\alpha)\exp\left[-\beta\rho(\|x(t)S\| + t)\right]\frac{1}{\|x(t)S\|^2}, \tag{7.16}$$

where ρ is the constant density of atmospheric particles, β is the extinction coefficient, $F(\alpha)$ is the phase function and α is the phase angle.

Observer on the ground : textures precomputation Dobashi et al. start by dividing $L_s(O)$ into a sum of small radiance quantities $\Delta L_s(x(k\ \Delta t))$, each corresponding to the total radiance in-scattered over one of the k segments of length Δt along the view ray, between positions $x(k\ \Delta t)$ and $x((k+1)\ \Delta t)$:

$$L_s(O) = \sum_{k=1}^{n} \Delta L_s(x(k\ \Delta t)) \tag{7.17}$$

Radiance $\Delta L_s(x(k\ \Delta t))$ in-scattered over a segment is then expressed as a product of two terms :

$$\Delta L_s(x(k\ \Delta t)) = f_h(x(k\ \Delta t)) \times f_l(x(k\ \Delta t)), \tag{7.18}$$

with :

$$f_h(x(k\ \Delta t)) = \int_{k\ \Delta t}^{(k+1)\ \Delta t} J(S)H(x(t))\mathrm{d}t \tag{7.19}$$

and :

$$f_l(x(k\ \Delta t)) = \int_{k\ \Delta t}^{(k+1)\ \Delta t} g(O,x(t))\mathrm{d}t \tag{7.20}$$

$f_h(x(k\ \Delta t))$ is a high frequency function which is evaluated at runtime, and $f_l(x(k\ \Delta t))$ is a low frequency term which can be partly precomputed and stored in a texture in order to speed-up the rendering process.

To decrease the complexity of $f_l(x(k\ \Delta t))$ so that it only depends on two parameters and therefore can be stored in a two-dimensional texture, the density of particles is assumed uniform, then $f_l(x(k\ \Delta t))$ can be simplified and reformulated as a product of two factors :

$$f_l(u, v) = c_k\ q(u, v), \tag{7.21}$$

where the first factor c_k only depends on the distance between O and $x(k\ \Delta t)$:

$$c_k = \exp\left[-\beta\rho(\|O\ x(k\ \Delta t)\|)\right] \tag{7.22}$$

The second factor, $q(u, v)$, is the quantity precomputed and stored in a texture. The first coordinate u is on an axis U parallel to the view ray and passing by the source, and the second coordinate v is on an axis V perpendicular to both the U axis and the view ray, and which intersects the view ray.

$q(u, v)$ is then given by :

$$q(u, v) = \int_u^{u+\Delta t} \rho F\left(\cos\left[\frac{-u'}{\sqrt{u'^2 + v^2}}\right]\right) \times \frac{\exp\left[-\beta\rho(\sqrt{u'^2 + v^2} + u' - u)\right]}{u'^2 + v^2}\mathrm{d}u' \tag{7.23}$$

Observer on the ground : rendering Atmospheric scattering is rendered using graphics hardware. To discretely evaluate $L_s(O)$, sampling planes are drawn in front of and parallels to the camera at sampled positions $x(t)$ along the view ray. On each plane is evaluated the radiance $L_{\text{gnd}}(x(t))$ in-scattered at this sample distance from the camera, and for each pixel of the image. The precomputations for $q(u, v)$ performed at the previous step are passed to the GPU as a texture, which is mapped in each plane to access the proper values. The visibility function $H(x(t))$ is evaluated using the shadow mapping algorithm [Cro77]. Sample radiances in-scattered on each plane perpendicular to the view ray are accumulated into the frame buffer.

Visualizing the atmosphere from outer space When the Earth is viewed from space, atmospheric particles are assumed to have a density decreasing exponentially with the altitude from the ground, and the Earth is considered perfectly spherical. Shadows from objects are not rendered. Two types of atmospheric particles are considered : air particles, which obey Rayleigh scattering, and aerosols, which obey Mie scattering.

Similarly to the previous case for an observer on the ground, precomputations are performed and textures are created, which are send to the GPU at the final rendering stage. The view ray is cut into k segments, and several quantities such as the radiance in-scattered along each segment for Rayleight scattering, and for Mie scattering as well are sampled into textures.

Whereas the atmosphere viewed from the ground was rendered by drawing planes in front of and parallels to the screen, in this case several concentric spheres of increasing radius are drawn around the Earth, and on which the computations are performed on the GPU. The total radiance received by each pixel is finally approximated by summing all k samples in the frame buffer.

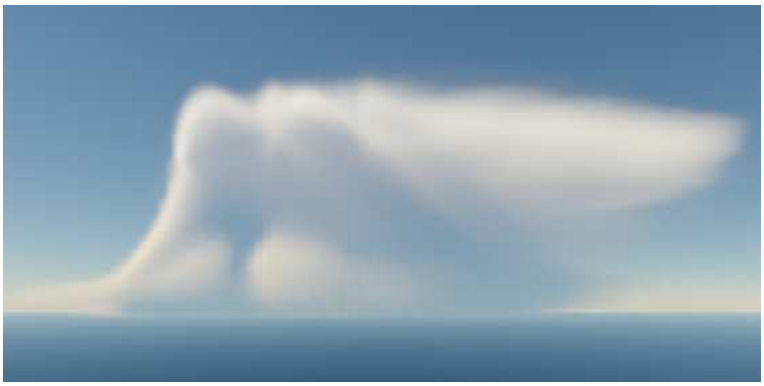

Figure 7.12: A low-albedo cloud illuminated using multiple-scattering and rendered using half-angle slicing. By Riley et al. [REK+04].

Finally, Dobashi et al. add clouds in the atmosphere when viewed from outer space, and which density is loaded from satellite images. By placing the camera at the sun's position and drawing concentric spheres intersecting the clouds layers, the clouds' density is mapped successively on all spheres and thus accumulated in the frame buffer. A texture is then generated containing the attenuation ratio at all points on the spheres. The final image is generated by using both the clouds textures and the attenuation textures obtained previously to compute the single-scattering illumination of cloud spheres on the GPU.

Conclusion This is one of the most comprehensive atmospheric illumination methods presented during the era of non-programmable GPUs. Dobashi et al. use hardware texturing and blending to reach interactive rates, through the classical algorithm based on volume slicing and the accumulation buffer. They efficiently reproduce most of the atmospheric scattering effects already considered in historical papers [NDN96] such as the change of color of the sky with the time, light beams generated by single-scattering and their illumination model also handles solid surfacic geometry, indeed the Earth's surface as well as other meshes. Although their cloud model, based on satellite images, is very simple, it is interesting to note the strategy employed to adapt the slicing process to the visualization of clouds from space : by applying a concentric spherical slicing around the Earth instead of slices parallel to the screen plane.

7.2.3.2 Interactive rendering of various light scattering effects in the atmosphere and clouds (K. Riley et al., 2004)

Overview Riley et al. [REK+04] are able to simulate a large number of different atmospheric effects in interactive time, using a specific model for each different medium particle. Since the majority of scattering effects in this work are caused by the precise distribution of peaks within the particles phase functions, the paper discusses a lot on how to best render such complex effects while keeping the computational cost reasonable.

Different types of particles The different types of particles rendered are :

- Atmospheric scattering effects by air molecules, for which Rayleigh scattering is used.

Figure 7.13: A cloud modeled with a homogeneous core and a heterogeneous envelope. The illumination approximates multiple-scattering using Premože's *most probable paths* [PAS03]. By Bouthors et al. [Bou08].

- Larger water particles such as rain drops and clouds for which Mie scattering is used. Peaks which can be observed at high angle values in their phase functions (around 130 degrees) lead to the visualization of rainbow effects. Peaks present when the scattering angle approaches 180 degrees generate glory effects, where rings of color can be observed in the clouds.

- Other types of large particles such as ice and snow, where the scattering probabilities only depend on the angle between the view ray and the incident light.

Rendering In this method, the Earth is assumed to be completely flat. The sky is rendered by analytically simulating single-scattering only, which is justified by the low probability of a scattering event in the air for great altitude angles. A model based on multiple-scattering phase functions is used for water and ice particles. Contrary to the sky, the density distribution of these particles is mapped on an uniform voxels grid, and rendering is achieved thanks to an extension of the half-angle slicing algorithm [KPH$^+$03] by Kniss et al.

The aerial perspective effect, which makes the sky appear blue during the day and become red at sunset, is also simulated.

Conclusion The most interesting aspect in this method is its ability to handle a large variety of cloud types, and to reproduce most of the optical effects observed under illumination by the sun. Although some papers try multiple-scattering through a mere isotropic spread of the neighborhood illumination [MHLH05] or assume perfect forward-scattering [Man06], Riley et al. improve the multiple-scattering algorithm used in the original half-angle slicing method by actually reproducing the iterative angular spread of the radiance as the direct lighting passes through each slice of the volume.

7.2.3.3 Interactive rendering of realistic clouds (A. Bouthors et al., 2007)

We finish this section with a specific but interesting model by Bouthors et al. [Bou08]. They illuminate and render clouds in interactive time by simulating multiple-scattering.

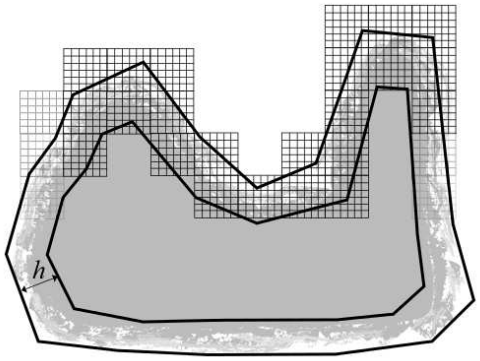

Figure 7.14: The cloud is modeled with a homogeneous center, surrounded by a hypertexture. From [Bou08].

The clouds are modeled in two parts : a homogeneous core, and a hypertexture which adds heterogeneous details on the bounding envelope (figure 7.14). They are mainly designed to be viewed from outside, even if the authors mention that their model can implicitely handle cases where the observer is inside a cloud. The technique provides good visual results.

Several orders of light scattering are simulated, using Mie's phase function. Single-scattering is calculated using an analytic version of the standard scattering equation [Kaj86], and multiple-scattering is calculated using an algorithm extending the method of the most probable paths [PAS03].

Multiple scattering is simulated by considering separately groups of four to eight scattering orders. For each group, the technique consists in two steps. First, the entry area on the envelope (called the *collector area*) where most of the light, which exits the cloud's volume towards the eye, is supposed to have entered the cloud in the first place. When this collector area is found, the manner in which this incoming light is most likely to be scattered through the cloud is estimated knowing how light is scattered at the same order within a sample rectangular volume.

At rendering, a shader is also applied on the coarse boundary mesh to give it a volumetric aspect, instead of looking like a solid surface, and the hypertexture containing the details is applied on the mesh.

Conclusion This method introduces a completely new way of modeling clouds, by associating both a homogeneous core and a heterogeneous perturbated enveloppe to add visual details. The approximation of the cloud's center by a homogeneous inner component whose optical density can then be evaluated analytically is very pertinent in the perspective of real-time, as a cloud's density is generally more important at its center and tend to decrease as we move towards its borders, so that in the case of large clouds viewed from inside, we can imagine that there would only be minor visual differences between a full-heterogeneous and a homogeneous core. Delimiting the cloud core using two solid meshes (an inner border delimiting the homogeneous core, plus an outer border delimiting the whole cloud including the hypertexture) enables to easily detect the entry and exit points on the GPU, while avoiding visual problems similar to bump-mapping, where a bumped surface betrays its flattiness when viewed from a parallel perspective.

Figure 7.15: Several explosions animated using fluid simulation. On the left : a 3D model is approximated by combining several 2D models using a Kolmogorov spectrum. On the right : the result using a true 3D model. By Rasmussen et al. [RNGF03].

7.3 Real-time smoke rendering and recent illumination techniques

In this last section, we discuss, in a first time, four techniques to render smaller media (in metric terms) such as smoke. We then finish with some recent works related to the propagation of light in volumes.

7.3.1 Real-time rendering of smoke

In the following methods, it is particularly interesting to compare the different strategies used for modeling the medium, which decides, then, how it is illuminated. We already began addressing this topic in section 6.2, with old methods choosing between either a regular grid of voxels, or particles like blobs/RBFs. These recent works go further and sometimes combine the two models, like in [ZRL+08] and [RZLG08]. We also review the popular *Fogshop* method [ZHG+07b], which is solely based on RBFs, but introduces an interesting approximation of pre-scattering optical depth that allows more precomputations.

7.3.1.1 3D smoke animation based on separate 2D fluid simulations (N. Rasmussen et al., 2003)

Rasmussen et al. [RNGF03] are able to simulate and render large scale smoke in interactive time by performing the underlying fluid simulation using a series of 2D velocity fields with a high resolution, instead of a full 3D grid. The smoke itself is modeled using a large number of particles, which are passively advected by the velocity fields. The computational cost is further reduced by do not accounting for interactions between particles, and the memory cost is reduced as well by not storing particles situated deep inside the medium, and therefore not visible.

The simulation of the fluid dynamics is based on the 2D incompressible Euler equations. Like in [FSJ01], the authors compensate the loss in interesting high-frequency swirling details in the veloc-

Figure 7.16: Multiple-scattering smoke modeled using, respectively from left to right, 100, 400 and 800 RBFs. By Zhou et al. [ZRL+08].

ity fields by using vorticity confinement. Moreover, buoyancy effects are simulated by introducing an external force to make the particles tend to go upwards, like in real smoke.

When all 2D velocity fields are assembled together, a Kolmogorov spectrum is used to help approximating a true 3D velocity field, by removing 2D artefacts and adding true 3D details. An interpolation is performed between all 2D velocity fields, in space and in time as well.

At the rendering step, all particles are mapped to a 3D grid perfectly aligned with the view frustum, and where each cell actually has the shape of a small frustum. This is an extremely interesting solution, when possible, because it allows a very fast grid traversal. The medium is illuminated by simulating single-scattering. In a first step, a ray-marching is performed from each light source's point of view throughout the grid, in order to obtain the attenuation of the direct lighting from this source at each voxel of the grid. Finally, a ray-marching is performed from the camera's point of view, both opacity and attenuated color are accumulated along the ray.

Conclusion The novel feature introduced by this method is the idea that it is possible to save a great amount of memory by assembling together a small number of independant 2D simulations instead of actually using a 3D velocity grid. At the time when this method was published, a typical simulation of smoke took around two minutes, as explained in the article. It would therefore be interesting to implement this method on modern programmable GPU's as real-time may certainly be reachable, since its use of separate 2D velocity fields makes it well adaptated for parallelization.

7.3.1.2 Real-time smoke rendering using compensated ray-marching (K. Zhou et al., 2008)

Zhou et al. [ZRL+08] render smoke modeled as a sum of RBFs in real-time using the GPU, under low-frequency environment lighting only. Single and multiple scattering of light are simulated.

The smoke's density distribution is first provided by the user under the form of a series of density fields in a volumetric voxels grid. As a preprocessing step, these density fields are decomposed into the combination of a low-frequency approximation using a RBF basis, with a residual field representing the high-frequency details missing to the RBF approximation to exactly recover the original density distribution. The residual field is then quantized on 8-bit values using perfect

spatial hashing [LH06] only before being sent to the GPU.

At runtime, the smoke is rendered using graphics hardware. First, the direct radiance reaching the center point of each particle is computed, but the details in the residual field are not yet considered at this stage. Multiple scattering is approximated by solving the diffusion equations at RBF centers (see [ZRL+08]).

Finally, the source radiance is approximated at every voxel in the medium's density grid by performing an interpolation between previously computed radiance values at RBF centers, while adding the local details from the compressed residual field. The volume is divided into a set of slices oriented perpendicularly to the view ray, and a ray-marching through the volume sums up the radiance in-scattered towards the camera to obtain the radiance reaching each pixel.

Algorithm 2 summarizes the visualization process, and highlights the combination of the low-frequency RBF approximation and a second compensatory term for the residual field. We note O the eye's position, P the nearest surface's position, $\tilde{\mathrm{T}}(A, B)$ the optical depth between A and B, $\tilde{\mathrm{D}}(\mathrm{x})$ low-frequency density field from the RBFs, and $\mathrm{R}(\mathrm{x})$ the residual field.

Algorithm 2 Rendering using the GPU [ZRL+08]

// Compute low-frequency approximation $\tilde{J}$ of direct lighting
for all RBF centers c_l **do**
  $\tilde{\mathrm{J}}(c_l) \leftarrow \tilde{\mathrm{J}}_{\mathrm{ss}}(c_l) + \tilde{\mathrm{J}}_{\mathrm{ms}}(c_l)$
end for
// Compute total radiance $L_m(x)$ along the view-ray
$L_m(O) \leftarrow 0$
for all sampled positions x along the view-ray **do**
  // Low-frequency approximation
  $L_m(O) \leftarrow L_m(O) + \tilde{\mathrm{T}}(x, P)\ \sigma_t\ \tilde{\mathrm{D}}(\mathrm{x})\ \tilde{\mathrm{J}}(x)$
  // Compensation for residual field
  $L_m(O) \leftarrow L_m(O) + \tilde{\mathrm{T}}(x, P)\ \sigma_t\ \mathrm{R}(\mathrm{x})\ \tilde{\mathrm{J}}(x)$
end for

Conclusion We already presented rendering algorithms which combine both a set of particles and a regular grid of voxels, by transforming the volume's density from one representation to the other, however this method involves such a conversion twice, in both ways. The medium's density is provided as a set of voxel grids, i.e. one for each frame sampled frame of the precomputed input smoke animation, from which is generated a set of RBFs where the direct lighting from the source is coarsely estimated. In a second time, prior to the final integration of the radiance in-scattered towards the camera, a grid is filled back by interpolating direct radiance quantities from the coarse illumination at each RBF center. Although this work itself does not consider the physical simulation necessary to animate the smoke, it can perfectly be plugged on the methods discussed above as a substitute to their only very basic rendering algorithm which was not, by the way, their main focus.

Figure 7.17: Single-scattering smoke rendered using the *Fogshop* method, and modeled using RBFs. By Zhou et al. [ZHG+07b].

7.3.1.3 Real-time single-scattering illumination of smoke modeled using RBFs (K. Zhou et al., 2007)

Overview The *Fogshop* method [ZHG+07b] is the most known work by Zhou et al., in which they propose an interesting optimization for single-scattering illumination of smoke modeled as a set of RBFs, using point light sources or environmental lighting. In comparison with [ZRL+08], no grid is used at all, and the algorithm works only directly on the RBFs. The radial function chosen for the RBFs is the Gaussian, the most common type of RBF.

Shadows by the medium onto itself and on the geometry are well rendered. However, it is important to note that shadows by the geometry are not computed, which removes a significant burden in order to reach real-time.

Single-scattering with a set of RBFs The medium is illuminated using a modified version of the single-scattering model, where the path taken by light is slightly changed in a way that allows to speed-up the evaluation of in-scattered radiance along the view ray.

Before we get deeper into details, let's quickly remind the base single-scattering model for media modeled as RBFs.

The density $\beta(x)$ of the medium at position x is written as :

$$\beta(x) = \beta_0 + \sum_{i=0}^{n} \beta_i(x), \tag{7.24}$$

where β_0 is the density of the homogeneous component of the medium, n is the total number of particles, and $\beta_i(x)$ is the density of the i^{th} particle composing the medium.

The radiance $L_a(O)$ due to the medium alone and reaching the observer at O is given by :

$$L_a(O) = \int_0^{d_r} \beta(x) f(x) dt, \tag{7.25}$$

where d_r is the distance between the observer at O and the nearest surface, i.e. the length of the view ray.

The radiance $f(x)$ in-scattered at point x on the view ray is expressed as :

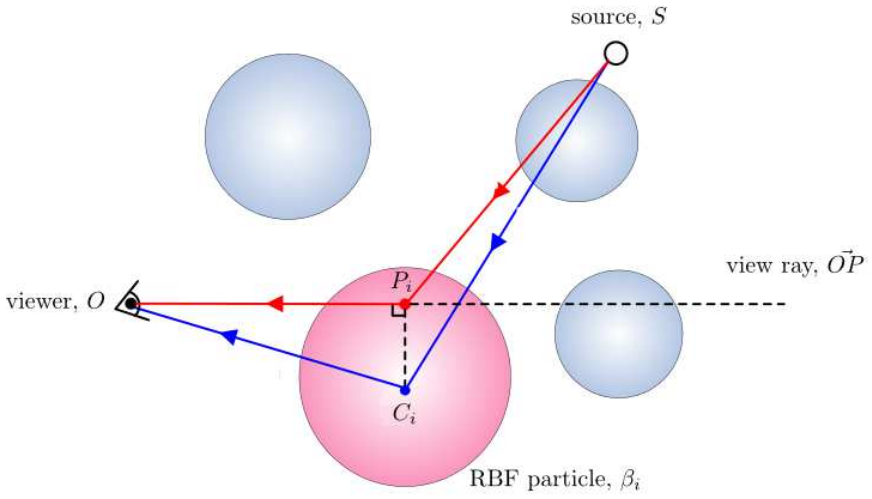

Figure 7.18: Approximation of optical depth computation when computing in-scattered radiance for particle β_i. The blue path is used for occlusion by all particles other than β_i, and the more accurate red path is used for particle β_i itself. Modified from fig. 3 in [ZHG+07b].

$$f(x) = k(\alpha(x))\frac{I_0}{d^2(x)}e^{-T(O,x)-T(x,S)}, \tag{7.26}$$

where $k(\alpha(x))$ is the scattering coefficient (with k the phase function), I_O is the intensity of the source at S, $d(x)$ is the distance between x and S, and $T(A,B) = \exp(-\|AB\|)$ is the optical depth between A and B.

Reformulation of the model Zhou et al. accelerate the computation of the radiance in-scattered at position x towards the viewer by simplifying the evaluation of the optical depth associated to, first, the direct lighting path from the source at s to the medium at x, and then, the post-scattering path from x to back to the observer at O.

We start with the full non-approximated model, and we begin by rewriting it in terms of the individual contribution of each RBF. Because the contribution of a sum of particles is equivalent to the sum of all individual contributions for each particle, we can write :

$$L_a(O) = \int_0^{d_r} \left(\beta_0 + \sum_{i=0}^{n} \beta_i(x)\right) f(x)dt \tag{7.27}$$

$$L_a(O) = \int_0^{d_r} \left(\sum_{i=0}^{n} \beta_i(x)\right) f(x)dt + \beta_0 \int_0^{d_r} f(x)dt \tag{7.28}$$

Because their respective variables i and t are independant, the integral along the view ray and the sum over all particles can be swapped :

$$L_a(O) = \sum_{i=0}^{n} \left(\int_0^{d_r} \beta_i(x) f(x) dt \right) + \beta_0 \int_0^{d_r} f(x) dt \tag{7.29}$$

We finally have :

$$L_a(O) = \sum_{i=0}^{n} L_i(O) + L_0(O) \tag{7.30}$$

with :

$$L_i(O) = \int_0^{d_r} \beta_i(x) f(x) dt \tag{7.31}$$

and :

$$L_0(O) = \beta_0 \int_0^{d_r} f(x) dt \tag{7.32}$$

From equations 7.26 and 7.31, we see that evaluating the radiance reflected by each individual particle requires computing the optical depth from S to x, then from x to O. To do this, all particles must be integrated along these two paths, which are different for each pixel, and therefore cannot be precomputed only once unless some simplification is performed on the model, which corresponds exactly the type of solution proposed by the authors.

Approximation of optical depth paths First, as x advances along the view ray, the position where the particle's density is evaluated is now dissociated from the position where the corresponding in-scattered radiance is computed (figure 7.18). Indeed, while the local density $\beta_i(x)$ remains evaluated normally at the moving position x, the scattering event associated to the contribution of particle i is now always evaluated at the fixed position of the projection of the particle's center C_i on the view ray, noted P_i.

We have :

$$L_i(O) \simeq f(P_i) \int_0^{d_r} \beta_i(x) dt \tag{7.33}$$

Second, to avoid re-computing again the accurate optical depth for each new ray, Zhou et al. suggest, when computing the in-scattered radiance $L_i(O)$ due to particle i, instead of evaluating $T(S, x)$ and $T(x, O)$ by actually integrating the density along these two connected paths, to approximate them by other paths that would have the advantage of being invariant with the pixel on the screen. Although the self-attenuation of the contribution of β_i due to particle i itself is computed using path passing by P_i, i.e. by integrating the density of β_i from S to P_i and then from P_i to O, the attenuation of the contribution of β_i due to all other RBFs is computed as if the scattering point was C_i, the center of particle β_i.

We finally have :

$$L_i(O) \simeq f^0(P_i) f^1(C_i) \tag{7.34}$$

with $f^0(P_i)$ the radiance in-scattered at the projection of the particle's center on the view ray, and only attenuated by β_i itself :

$$f^0(P_i) = \frac{1}{4\pi}\frac{I_0}{\|S\ P_i\|^2 + h^2}\mathrm{e}^{-T_i(S,P_i)-T_i(O,P_i)+T_i(S,C_i)+T_i(O,C_i)} \tag{7.35}$$

and the factor $f^1(C_i)$ corresponding to the rest of the attenuation by other particles :

$$f^1(C_i) = \mathrm{e}^{-T_i(S,C_i)-T_i(O,P_i)} \tag{7.36}$$

The computation of the homogeneous component $L_0(O)$ of the medium is also simplified, in a similar fashion, which we will not detail here (please refer to [ZHG+07b] for details). The self-attenuation by the medium itself is determined from the accurate distance traveled by the light rays, whereas, the attenuation by the heterogeneous particles is reduced to a simple direct path from the camera to the light source.

Illumination of surfaces with the PSF Beside the optimized illumination of the set of RBF particles, Zhou et al. also illuminate the geometry of the scene by first taking into account shadows caused by the medium's presence between the source at S and the surface at P, but also approximate radiance incoming on the surfaces due to out-scattering towards their direction.

Shadows caused by the medium are due to pure attenuation from S to P and can be computed normally based on the optical depth $T(S,P)$, resulting in the attenuation factor $\mathrm{e}^{-T(S,P)}$.

The post-scattering illumination of surfaces would be too complex to compute the same way as the illumination reaching each pixel, i.e. by evaluating $L(P)$ like if the pixel was at P. Instead, Zhou et al. use the point spread function (PSF) to estimate how much the light from the source would be scattered when reaching P based on the distance from S to P.

The light $L_P^{\text{in-ss}}$ reaching each surface at P is approximated by :

$$L_P^{\text{in-ss}} = L_P^{\text{in}} * PSF \tag{7.37}$$

This simply consists in performing a convolution between the radiance L_P^{in} incoming at P without considering the medium, and the point spread function

Rendering Due to the approximations described above, these precomputations now play a major role in the whole rendering process, using programmable GPUs and shaders. Rendering is achieved through several successive stages, each stage involving a rendering of the particles and the scene's geometry from a different point of view, the results being stored using temporary buffers. For more specific details about the algorithm, please refer to [ZHG+07b].

In a nutshell, a depth map is first computed from the camera's point of view and in its viewing direction, and then from each light source towards six directions to sample view rays over the whole sphere (i.e. a cube map). Only the geometry is rendered in this first step. Then, the optical depth is accumulated into another cube map for each light source, where both the geometry and RBFs are rendered, by drawing their respective bounding sphere and using a shader to compute the actual traversed density. The geometry alone is again rendered from the camera's point of view and the light $L_P^{\text{in-ss}}$ scattered by the medium and illuminating the surfaces is computed for each pixel. Then, the single-ray optical depths are integrated from the camera to the center of each RBF, from the camera to each source, and from each source to the center of each RBF.

Figure 7.19: Light enters the room by the window which casts shadows on single-scattering smoke. By Ren et al. [RZLG08].

Finally, after all optical depth values and the illumination of surfaces have been precomputed, the radiance ultimately reaching each pixel can be summed. The total light reaching a pixel is composed of the appearance of the medium itself, to which is added the appearance of the surface, attenuated by the medium all way long from the camera to the surface. Indeed, the light reflected by the surface is not further scattered by the medium like it would be in the real world, but only attenuated by it.

This process is summarized in algorithm 3.

Conclusion Unlike similar works [ZRL+08] using both particles and a grid of voxels, this method only works on the same set of RBFs from precomputations to rendering. The advantage of this method might be, first, its simplicity, since it only requires using simple functionalities of the GPU pipeline, with a few shaders to evaluate the gaussians, and some intermediary renderings into temporary textures.

The reformulation of the model is very sound. We start with the problematic of rendering the medium as a whole, whose particles are all shadowing each other and can be situated anywhere in the scene, making precomputations tricky. At the end, the result is a sum of small individual models, where the illumination of each individual particle is considered as a separate problem, making it possible to fully exploit the particle nature of the medium.

However, this work do not consider occlusions by the geometry, which is one of the important challenges for the development of a real-time unified model which would handle both arbitrary media as well as solid objects. One of the advantages of the use of a function basis, like a set of RBFs, is the possibility to model large smooth media using only a handful of particles, instead of a regular grid where large voxels would lead to strong visual artefacts with sharp transitions, but this advantage is negated here by the optimizations which need the particles to be small for the approximations to remain acceptable.

Algorithm 3 Rendering process [ZHG+07b]

```
Place camera at eye's position O
Compute depth map d_O
for all light sources s do
  Place camera at S_s the position of source s
  Render geometry and compute depth cube map d_s
  Accumulate optical depth in cube map T(S_s, P) around S_s
end for
Place camera at eye's position O
Compute surface illumination L(P)
for all RBFs i do
  Compute T(O, C_i)
end for
for all light sources s do
  Compute T(O, S_s)
  for all RBF i do
    Compute T(S_s, C_i)
  end for
end for
Compute appearance L_a(O) of the medium alone
Compute optical depth T(O, P) from the eye's position
L(O) ← L_a(O) + exp(−T(O, P)) L(P)
```

7.3.1.4 Real-time single-scattering illumination of smoke using gradient-based interpolation (Z. Ren et al., 2008)

Ren et al. [RZLG08] also illuminate heterogeneous single-scattering smoke in real-time, and their model enables to reproduce a variety of lighting effects, such as glows, light shafts and volumetric shadows. The density function is first provided as a grid of density voxels, and then approximated by a sum of Gaussians. Some precomputations are performed, but the main geometry and lighting configuration can change at runtime.

At runtime, the following four-step algorithm is used to illuminate the medium and compute the radiance reaching each pixel (Refer to algorithm 4 for details) :

1. A number of sample points are generated and dynamically distributed over the medium. The initial set of samples simply includes the center point of all RBFs, and is recursively refined if needed.

2. The radiance directly reaching each sample point is computed, together with the gradient of its radiance.

3. The radiance reaching every other points in the volume is approximated thanks to a gradient-based interpolation, taking into account both the radiance level and its gradient at the closests sample points, for greater accuracy.

4. The medium is rendered using a classical integration of the in-scattered radiance along the view ray.

The rendering method is detailed in algorithm 4.

The sampling method is recursively adaptative, indeed new samples are automatically inserted in zones containing more lighting details. This is achieved by detecting local shading errors with a sphere around a sample point, and then generating new sample points inside that sphere until these errors are compensated. Algorithm 5 explains this dynamic sampling process in details.

Algorithm 4 Gradient-based illumination [RZLG08]

// Sample sparse points x_j and compute lighting and gradient there
Dynamically generate sample points $\{x_j\}$
for all sample points x_j **do**
  Compute direct lighting L_{x_j}
  Compute direct lighting gradient ∇L_{x_j}
end for
// Interpolate lighting at other points x in the medium
Align camera with the Z-axis of the volume
Switch to parallel projection
for all XY-slices s of the volume **do**
  Build list Q_s of all samples x_j whose bounding quad crosses slice s
  // Interpolate between samples
  for all pixels x to interpolate **do**
    $L_x \leftarrow 0$ *// Weighted radiances accumulator*
    $W_x \leftarrow 0$ *// Weights accumulator*
    for all bounding quads q intersecting slice s containing x **do**
      Draw bounding quad q
      $W_j = R_j / \|x - x_j\|$
      // Using a shader, output :
      $L_x \leftarrow L_x + W_j(L_{x_j} + (x - x_j)\nabla L_{x_j})$
      $W_x \leftarrow W_x + W_j$
    end for
    $L_x \leftarrow L_x / W_x$
  end for
end for
// Final ray-marching from the camera
Compute final radiance $L(O)$ with a ray-marching along the view-ray

Conclusion Ren et al. present this method as an improvement to their similar previous RBF-based smoke rendering techniques [ZHG+07b, ZRL+08], by enriching it with two new features. Like in the Fogshop method, the L-BFGS-B minimizer is used to generate this set of RBFs from the density grid, and the direct lighting is computed at the center of each particle. The first difference with previous methods takes place after the construction of the RBF samples, when interpolating

Algorithm 5 Dynamic sampling [RZLG08]

```
Generate a first set of samples Q^0 = {c_j} with RBFs centers
for all samples c_j in Q^0 do
  Compute shading error E_j around c_j
  if E_j > ε then
    Allocate empty set Q_j^1 for new samples
    Intersect sphere around c_j with a thinner grid G^1
    for all vertex q from G^1 which lie within sphere around c_j do
      Affect radius equal to resolution of G^1
      Push q into Q_j^1
    end for
    Repeat process with samples of Q_j^1 while needed
  end if
end for
```

the direct radiance at each voxel of the grid using gradient-based interpolation and splatting, a more accurate interpolation technique than the simple interpolation from the particles closest to the interpolated voxel. The second and major improvement lies in the use of their recursive sample splitting algorithm, which enables to dynamically reduce visual artefacts by adding new RBF particles to improve the accuracy of the interpolation.

7.3.2 Recent illumination techniques using the GPU

We finish this state of the art with three recent works. The first one, [Kap09] interests us particularly since it was our main source of inspiration for our work on the propagation of occlusions in a volume. It introduces the concept of Light Propagation Volumes, and use it for the propagation of indirect lighting between diffuse surfaces. The second one, [Fat09] presents the Light Propagation Maps, based on a completely different strategy. Light is propagated in a volume similar to LPVs, but the algorithm itself, instead of directly working on the volume containing the result, uses a set of 2D arrays of light rays with varying directions, along which the radiance is iteratively scattered and the volume updated. Finally, in [CBDJ11] is presented a new approach for the simulation of shadows in a scene featuring both solid geometry and participating media.

7.3.2.1 Light propagation volumes (A. Kaplanyan, 2009)

Note : This technique does not handle smoke or any kind of participating medium. Its purpose is to compute indirect illumination of solid surfaces.

Overview One of the techniques that will particularly interest us is described in [Kap09], and is known as *light propagation volumes* (LPV). Developed as part of Crytek®'s CryEngine™3 video game engine, it comes as a novel solution to coarsely approximate full-diffuse indirect illumination on solid objects.

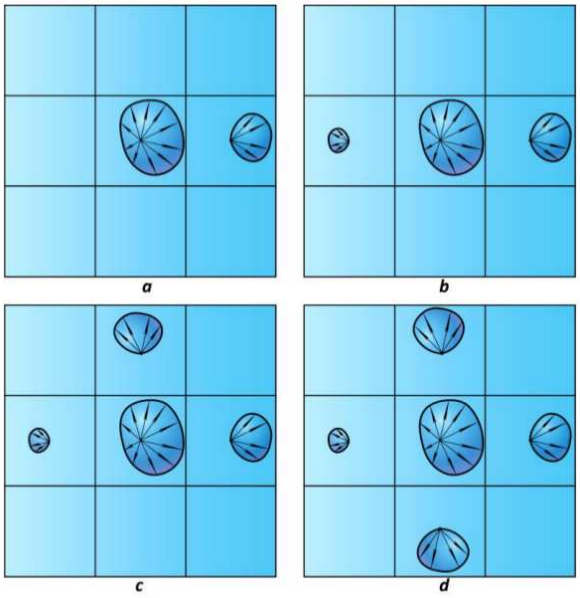

Figure 7.20: Iterative radiance propagation towards four neighbouring cells. From [Kap09].

The main advantage of this method is the absence of any kind of precomputation, allowing a dynamic geometry, camera position and lighting. As Kaplanyan explains, one bounce of indirect lighting is already sufficient to obtain a fully satisfying degree of visual realism.

The method Light propagation volumes are only used for the second lighting step, i.e. the indirect illumination, after direct lighting is computed and the indirect diffusely reflected radiance is ready to exit the surfaces. The first step consists in computing the direct illumination of surfaces using the reflective shadow maps (RSM) method [DS05], which results in a set of *virtual point sources* (VPL) being deposited on all lit surfaces. All these secondary light sources are then filtered based on importance sampling, so that their number can be reduced to only keep the most significative VPLs.

The angular distribution of the radiance emitted by each VPL is first represented using spherical harmonics. All VPLs are then injected into the cells of a light propagation volume, which has the form of a volumetric grid, implemented using a 3D texture.

The main process consists in a step-by-step propagation of the radiance injected from the VPLs into the LPV, using a 6-neighborhood scheme. Each grid cell contains four spherical harmonics coefficients, i.e. two levels, representing the amount of radiance escaping the cell towards all six neighbours. The propagation stops when the distribution of the indirect lighting throughout the LPV has become stable.

The fourth and ultimate step simply consists in the visualization of the illuminated geometry. Since we are not in the presence of a scattering medium, this step becomes even simpler, as all the radiance has already been propagated in the whole scene space, together with its angular distribution. A simple texture mapping on all surfaces covered by the LPV will suffice to achieve the rendering of the indirect illumination.

Distributing VPLs across the scene The whole LPV illumination process begins with the creation of a reflective shadow map (for more details, please refer to [Kap09]). By analogy to the classical *shadow mapping* [Wil78, RGK+08], obtaining the RSM involves rendering the scene, which is assumed to only contain diffuse surfaces, from the point of view of the light source. At the implementation level, a couple of shaders is used to output in multiple render targets all informations needed with every pixel :

- The depth of the fragment, i.e. the content of a standard shadow map
- The position of the fragment in the global world space
- The normal to the surface
- The RGB intensity of the radiance diffusely reflected by the surface

Every pixel in the RSM is considered as a secondary point light source on which is based the indirect illumination. In the context of LPVs, an importance-driven downsampling is then applied on the RSM to reduce the number of these secondary light sources.

Injecting VPLs into the LPV At this step, a simple list of sparse VPLs has been created, and now needs to be injected into the LPV, implemented as a 3D texture. This is achieved thanks to *point-based rendering* [BHZK05], involving rendering each individual VPL as a small surfel, on which is applied a shader whose role is to write the VPL's properties in the proper corresponding cell in the LPV, indeed the cell in which the surfel falls. The sample single LPV stores the indirect radiance of all VPLs, whose orientation differs, the form under which the radiance is injected in the LPV must consider both an intensity and a lighting direction. Although those informations are already stored explicitely in the reflective shadow map, several VPLs with different properties can be injected into the same LPV cell, this is why Kaplanyan chooses to represent the content of the LPV in a basis of spherical harmonics (SH), using two levels, i.e. four coefficients. The main advantage of a spherical harmonics basis is the fact that even if the radiance and orientation of VPLs are distinct, injecting such two separate VPLs into the same cell only requires summing together the four respective coefficients.

Spherical harmonics coefficients If $\vec{n}(n_x, n_y, n_z)$ is the normal vector to the surfel, then its decomposition c in two levels of SH is given by :

$$c = (c_0, c_1, c_2, c_3) \tag{7.38}$$

$$c_0 = \frac{1}{2\sqrt{\pi}} \qquad c_1 = \frac{\sqrt{3}n_y}{2\sqrt{\pi}} \qquad c_2 = \frac{\sqrt{3}n_z}{2\sqrt{\pi}} \qquad c_3 = \frac{\sqrt{3}n_x}{2\sqrt{\pi}}, \tag{7.39}$$

where c_0 is the unique coefficient for the first level of SH, and c_1, c_2, c_3 are the coefficients for the second level.

For more details, please refer to [Slo08].

Injected RGB radiance For the moment, we did not consider any color information, as the SH coefficients c are themselves not composed of color channels, but are instead only used for the SH decomposition. To compute the actual RGB color components (c_r c_g c_b) corresponding to the radiance reflected by the surfel, we must take into account the material properties of the surface, the intensity of the source, the angle between the normal and the light direction, and the importance (weight) of the surfel, without omitting the coefficients c computed above. Similarly to c which represents a vector of four SH coefficients, each c_r, c_g and c_b is also composed of four SH values. We have :

$$\begin{pmatrix} c_r \\ c_g \\ c_b \end{pmatrix} = c^T \left(I_L \; A_S \; I_S \; W_S \right), \tag{7.40}$$

where I_L is the instensity of the source, A_S is the albedo of the surface, I_S is the diffuse reflection coefficient, equal to $I_S = (\vec{n} \cdot \vec{l})_+$, the positively clamped dot product between the normal to the surfel $\vec{n}$ and the lighting direction $\vec{l}$, and W_S is the weight of the surfel.

Propagation At this step of the illumination process, the radiance reflected on the surfaces after the direct lighting, as three RGB components, themselves decomposed in four SH coefficients, have been injected in the light propagation volume. The propagation process is rather simple, and is performed step-by-step, following a 6-neighborhood scheme (illustrated for the 2D case in figure 7.20).

At a glance At each propagation step, all cells in the LPV which are directly connected to at least one neighbouring cell containing already propagated radiance are processed, and this includes empty cells as well as those already filled during the previous steps. When a cell is processed, it gathers the radiance exiting towards its direction from all six neighbours, and simply sums up the SH vectors resulting from these six gatherings.

This radiance gathering operation can easily be implemented on the GPU, using vertex/fragment shaders. A common strategy for this type of algorithm consists in using two different LPV textures, alternatively used for reading or writing the radiance gathered at the current step, using *multiple render targets*. To achieve one step of propagation, the algorithm advances slice by slice along the depth axis and processes each single slice separately. One slice is used for reading the neighbouring cells radiance, as propagated until the previous step, and the other is used for writing the sum of the gathered radiance on each voxel processed this way by the fragment shader.

This process is described in algorithm 6.

Gathering radiance from neighbours The four SH coefficients stored in each cell of the LPV model the whole distribution of the radiance as it exits the cell. This representation allows evaluating, of course, the radiance emitted in a punctual particular direction, by means of a classical evaluation of the SH basis with the neighbour's coefficients. However, punctual samples are of no use for our gathering operation, and instead, what we would ideally need, are the SH coefficients for the same radiance distribution, but where only the proper angular range corresponding to this neighbour-to-neighbour transmission is kept, the rest being set to zero.

This angular window filtering can be achieved by multiplying, on the one hand, the all-directions original SH coefficients read in the neighbour cell, with four special SH coefficients describing a window filter, whose shape represents a lobe. Unrotated, this filtering lobe is aligned with the Z-axis, and can be described with two zonal harmonics coefficients : $[0.25, 0.5]$[1]. However, it must be rotated to fit the six possible gathering directions.

The radiance $L_{A\to B}$ transmitted from cell A to the gathering cell B is given by :

$$L_{A\to B} = a \times f, \tag{7.41}$$

where a is the SH coefficients vector from cell A, and f is the SH coefficients vector of the angular filter, given by :

$$f = \begin{cases} f_0 = 0.25 \\ f_1 = 0.5 \times \sqrt{1 - d_z^2} \times d'_y \\ f_2 = -0.5 \times d_z \\ f_3 = 0.5 \times \sqrt{1 - d_z^2} \times d'_x \end{cases}, \tag{7.42}$$

where $\vec{d} = A - B$ is the gathering direction, and $\vec{d'}$ its normalized projection along the XY plane :

$$\vec{d'} \begin{pmatrix} \frac{d_x}{\sqrt{d_x^2 + d_y^2}} \\ \frac{d_y}{\sqrt{d_x^2 + d_y^2}} \\ 0 \end{pmatrix} \tag{7.43}$$

The final values to write in the LPV for each processed cell are the simple sum of the six SH vectors expressing the radiance gathered from all six neighbouring cells. Because the radiance actually possesses three color components, a total of eighteen gathering operations are required for each cell.

Please refer to algorithm 7 for details.

Rendering the LPV To illuminate the scene based on the light propagation volume in which the indirect radiance reflected from the diffuse surfaces has been propagated, one must simply map the LPV texture on the geometry of the scene. Indeed, the resulting indirect radiance reaching each surface of the scene is precisely the quantity which was propagated, and is now contained in the propagation volume. The radiance can now directly be fetched from the volumetric texture.

After the propagation, it is impossible to know where the light contained in a voxel originated exactly, since it is most often a mix of the different radiances propagated from a large number of VPLs. As a result, it appears difficult to apply any additional lighting model (Phong, etc.) when shading the surfaces. However, doing so would only be appropriate when dealing with direct lighting.

[1]The zonal harmonics [a b] correspond to the spherical harmonics [a 0.0 b 0.0].

Algorithm 6 Propagating radiance in the LPV [Kap09]

// Propagation initialization
// We note lpv1 *the input LPV texture*
Align camera with the Z-axis of the LPV
Setup parallel projection
Set framebuffer dimensions to match those of the LPV XY-slices
Allocate **lpv2** as a second LPV texture
// Main propagation process
for all propagation steps by increment of 2 **do**
  // Even propagation step
  Bind 3D texture **lpv1** as shader input
  Bind 3D texture **lpv2** as shader output
  for all slices k **do**
    Draw fullscreen quad cutting slice s
    for all pixels/voxels $v_{i,j,k}$ in the grid slice **do**
      $v_{i,j,k} \leftarrow$ black
      $v_{i,j,k} \leftarrow v_{i,j,k} + L_{(i-1,j,k)\rightarrow(i,j,k)}$
      $v_{i,j,k} \leftarrow v_{i,j,k} + L_{(i+1,j,k)\rightarrow(i,j,k)}$
      $v_{i,j,k} \leftarrow v_{i,j,k} + L_{(i,j-1,k)\rightarrow(i,j,k)}$
      $v_{i,j,k} \leftarrow v_{i,j,k} + L_{(i,j+1,k)\rightarrow(i,j,k)}$
      $v_{i,j,k} \leftarrow v_{i,j,k} + L_{(i,j,k-1)\rightarrow(i,j,k)}$
      $v_{i,j,k} \leftarrow v_{i,j,k} + L_{(i,j,k+1)\rightarrow(i,j,k)}$
      Update radiance of voxel $v_{i,j,k}$ in the framebuffer
    end for
  end for
  // Odd propagation step
  Bind 3D texture **lpv2** as shader input
  Bind 3D texture **lpv1** as shader output
  [...]
  Repeat the slice-by-slice propagation scheme described above
  [...]
end for

Conclusion This indirect illumination technique is very interesting, since it is particularly adapted for running on the GPU. The propagation process itself requires a lot of small computations, but because it is the exact same gathering process which is executed on each voxel and at each iteration, [Kap09] manages to minimize non-GPU friendly operations such as loops, branches and tests. Moreover, this propagation process can be efficiently implemented both using GPGPU frameworks such as CUDA, or programmable shaders, by means of "ping-pong rendering" between two textures, as discussed in chapter 12. Because the GPU memory accesses for both reading and writing voxel values are spatially coherent, as we could experience ourselves in our own work, the whole propagation process proves remarkably fast, even for important volumes.

Algorithm 7 Gathering radiance from cell A towards cell B [Kap09]

// Compute gathering directions
$\vec{d} \leftarrow (B_x - A_x, B_y - A_y, B_z - A_z)$
$\vec{d'} \leftarrow (d_x/\sqrt{d_x^2 + d_y^2}, d_y/\sqrt{d_x^2 + d_y^2}, 0.0)$
// Compute filtering lobe's SH coefficients
$f_0 \leftarrow 0.25$
$f_1 \leftarrow 0.5 \times \sqrt{1 - d_z^2} \times d'_y$
$f_2 \leftarrow -0.5 \times d_z$
$f_3 \leftarrow 0.5 \times \sqrt{1 - d_z^2} \times d'_x$
// Update cell B's radiance
$L_{A \to B} \leftarrow L_A \times f$
$L_B \leftarrow L_B + L_{A \to B}$

7.3.2.2 Participating media illumination using light propagation maps (R. Fattal, 2009)

Note : This method does not run in real-time at the time of its publication, but it is nevertheless worth to be mentioned, as a technique for the propagation of light in a volume.

Overview In this work, Fattal [Fat09] develops a new method to propagate light in volumes such as smoke, as well as more solid volumetric materials like marble. The previously discussed work by Kaplanyan [Kap09] was very representative of the methods based on finite elements to approximate the illumination throughout a scene using a regular 3D grid, by iteratively propagating the radiance from cell to cell.

This method adopts a different approach, and belongs to the category of the discrete ordinates methods, inaugurated by Chandrasekhar [Cha50], for which the 3D grid is discretized in both space and orientation, and is used to iteratively solve the radiative transfer equation. It can also be classified in the Lattice-Boltzmann methods, since instead of working directly on the 3D grid of voxels and propagating energy from neighbour to neighbour, it works with indirect sets of 2D maps of light rays with similar directions that sweep through the separate 3D grid along different directions to propagate the radiance.

The light propagation map The whole workspace is discretized in terms of position and angle. Fattal uses six light propagation maps (LPM), each one associated to an axis (X^-, X^+, Y^-, Y^+, Z^- and Z^+), which indicates the dominant coordinate in the rays' normalized direction, and which serves as the axis of reference to which is attached the ray's parameter.

For instance, all rays grouped in the Z^+ map are defined as follows :

$$R^n_{r,s}(z) = \left(x_{r,s} + z \cdot \frac{\omega^n_x}{\omega^n_z},\ y_{r,s} + z \cdot \frac{\omega^n_y}{\omega^n_z},\ z \right), \tag{7.44}$$

where n is the index of the direction on the discretized sphere S^2, r and s are the 2D coordinates of the ray in the map, $x_{r,s}$, $y_{r,s}$ and z are its coordinates in the scene (z being passed directly as the

parameter to move along for all rays in map Z^+), and $(\omega_x^n, \omega_y^n, \omega_z^n)$ are the coordinates of direction n.

Radiance evolution along a ray At each step, the evolution of the intensity of ray $R_{r,s}^n$ can be expressed as :

$$\omega_z^n \frac{\mathrm{d}}{\mathrm{d}z} L_{r,s}^n(z) = -(k(R_{r,s}^n(z), \omega^n) + \sigma(R_{r,s}^n(z)))L_{r,s}^n(z) + u(R_{r,s}^n(z), \omega^n), \tag{7.45}$$

where $L_{r,s}^n(z)$ is the current radiance on the ray, $k(R_{r,s}^n(z), \omega^n)$ is the absorption coefficient, $\sigma(R_{r,s}^n(z))$ is the scattering coefficient, and $u(R_{r,s}^n(z), \omega^n)$ is the radiance already present in the grid at position $R_{r,s}^n(z)$ and going in direction ω^n, due to emission and scattering at previous steps.

Algorithm In this method, light sources are not placed and manipulated as such, but are instead injected into the 3D grid, as a light-emitting cell or group of cells. To illuminate a scene from above, for example, some cells on the top of the volume would need to have a non-null emission, whether their density equals zero (just a light sources, no medium or object) or not (the source lie within something an object or a participating medium). After the sources are injected in the volume as its initial state, the algorithm propagates emitted light indifferently in the air or inside non-opaque matter, the scattering coefficient associated to each cell determining the scattering behaviour of light rays.

Equation 7.45 expresses how the radiance carried by ray $R_{r,s}^n$ evolves along the successive steps, as we iteratively move along the ray by increasing z, integrate these changes and write the result at each cell (i, j, k) in the 3D grid. At each integration step, we do not advance on each ray regularly, but by successive intersections with the grid planes, such that at each step is computed the integral along the exact small segment of the ray that crosses a new grid cell.

The grid cells stores several quantities, which are updated at each step :

- $I_{i,j,k}^m$ accumulates the radiance to construct the final result for each cell
- $K_{i,j,k}^m$, the local absorption coefficient
- $S_{i,j,k}^m$, the local scattering coefficient
- $E_{i,j,k}^m$, the local medium emission
- $U_{i,j,k}^m$, the radiance arrived on the cell due to previous steps, and which still has to be propagated

Algorithm 8 gives a general idea of the propagation scheme (refer to [Fat09] for details).

At the initialization step, both the final result accumulator $I_{i,j,k}^m$ and the temporary unscattered radiance buffer $U_{i,j,k}^m$ are initialized with the emission $E_{i,j,k}^m$. The algorithms then repeatedly sweeps the grid with all six LPMs while all radiance still has not been fully propagated, i.e. when some unscattered radiance remains temporarily stored in $U_{i,j,k}^m$. For each map ray, we advance cell-by-cell, where both the newly in-scattered radiance in direction m and the portion of the radiance already present in $U_{i,j,k}^m$ which can be propagated by the current sweep are accumulated in $L_{r,s}^n$, i.e. on the current ray. Finally, the radiance out-scattered from the current cell in all directions is updated on both the final result $I_{i,j,k}^m$ and in $U_{i,j,k}^m$.

Algorithm 8 Radiance propagation using LPMs [Fat09]

```
Init I^m_{i,j,k} with E^m_{i,j,k} for all cells
Init U^m_{i,j,k} with E^m_{i,j,k} for all cells
// Keep propagating while there remains unpropagated radiance
while max_{m,i,j,k}(|U^m_{i,j,k}|) ≥ ε do
  // Do a complete sweep along all LPMs
  for all map Ω ∈ {X⁻, X⁺, Y⁻, Y⁺, Z⁻, Z⁺} do
    // Integrate along all rays in map Ω
    for all rays in map Ω do
      for all ray segments do
        Update L^n_{r,s} with in-scattered radiance along segment
        // Update grid states for next sweeps
        for all directions m do
          Update final result in I^m_{i,j,k}
          Store in U^m_{i,j,k} radiance non-scatterable at current sweep
        end for
      end for
    end for
  end for
end while
```

Conclusion This method is interesting since it presents a great potential for optimization, what was attempted very recently in [GPC+12]. This strategy of not working directly on the grid which accumulates the final result, but instead propagating light along separate rays independent from the gridwise neighbourhood relations between cells provides a true solution for the two main problems of common finite element methods, i.e. the oversmoothing of radiance (and shadows, by the way) after a certain number of propagation steps, as well as the difficulty for a single square cell set as a source to emit light uniformly in all directions. Unfortunately, this method itself is far from even interactive time, but the idea of successively sweeping a volume along different directions remains interesting.

7.3.2.3 Real-time volumetric shadows using 1D min-max mipmaps (J. Chen et al., 2011)

Overview In this work, Chen et al. [CBDJ11] present a method to compute shadows in single-scattering media, based on a modification of the shadow mapping algorithm featuring an epipolar rectification of the shadow map, allowing the subsequent construction of a graph used to quickly determine which portions of a given view ray receive direct lighting, and which portions are shadowed.

The rectified shadow map What the standard shadow map stores is, for each light ray in the source's screen coordinates, the distance below which a point is illuminated, and above which a point is shadowed. Actually, the less trivial step in the shadow mapping consists in mapping

Figure 7.21: Light entering by a window in a church. From [CBDJ11].

a point in the scene from the viewer's point of view towards the source's coordinate system, to know which light ray (i.e. which associated pixel of the shadow map) intersects the same given point. This is due to the fact that there is no direct relationship at all between the shadow map's coordinate system and the camera's point of view.

Epipolar geometry is commonly used in the domain of computer vision and stereoscopy, for example to rectify a couple of stereoscopic images so that all points common to both images are situated on the same row of pixels. Following a similar idea, an epipolar rectification is applied on the shadow map to deform it in such a manner that (figure 7.22) :

- The rectified shadow map still stores the same information as the stancdard shadow map : each pixel corresponds to a light ray, the values indicates (under a different form) the depth at which a surface is intersected.
- Each 1D slice (row in the rectified shadow map) is associated to a series of light rays forming the same plane containing both positions of the viewer and the source. Therefore, all light rays which project on the same half-line (whose origin is the viewer's position) on the source's screen plane belong to a common slice in the rectified shadow map. On the ordinates, these slices are indexed with a coordinate α, mainly related to the angle in polar coordinates of their projection on the source's screen.
- On the abcissa, a coordinate γ allows to navigate between all light rays belonging to the same slice α. This coordinate varies with the angle between the light ray and the direction from the source to the viewer.
- Instead of storing a direct depth, to each view ray (α, γ) is associated a value β, which is the angle that the view ray intersecting the same point on the surface forms with this view ray.

With the standard shadow map, we are not provided any direct information regarding which view ray intersects the surface illuminated by a given light ray. The main advantage of epipolar planes is that they contain both a series of light rays and their associated series of view rays which may intersect them on lit surfaces. The domain of study being reduced to a plane common to both

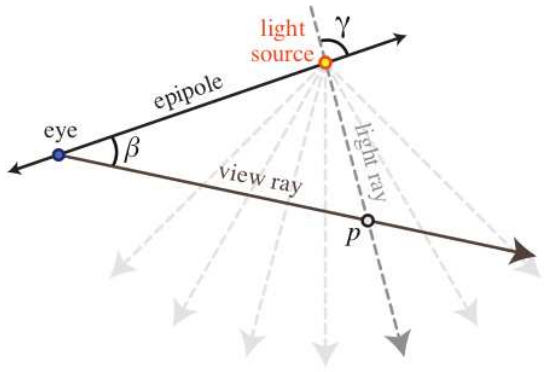

Figure 7.22: An epipolar slice and the epipolar coordinates. Light rays are indexed with angle γ. View rays are represented with angle β. From [CBDJ11].

types of rays, provided with a reference axis (linking the viewer and the source), the view rays can be represented with a single angle β, that their direction form with this axis.

When β is near π, its maximum value, the view ray has a direction close to that of the light, and intersects at a great distance, which tends to infinity. As β decreases, so does the distance from the source to the intersection. Because of this similarity, it is possible to replace the depths in the shadow map with angles β, with the advantage that they directly point to a particular view ray.

Testing whether a surface is shadowed or not is now much easier. The coordinates (α, γ) in the shadow map are computed based on the direction from the source to the surface, and angle β is extracted. If the angle of the view ray is lower than β, the surface is shadowed, and if both angles are equal, the surface is lit. In the presence of a participating media, which is not surfacic and therefore not taken into account in the shadow map, the view ray's angle can be strictly greater than the value in the shadow map, which also indicates that it receives light.

The min-max mipmap trees Computing the illumination of a single scattering medium involves evaluating the integral of the radiance in-scattered along each view ray. At each integration step, the shadow map must be accessed to determine if the current position in the scene is lit or shadowed. With regards to the shadow map, because the view ray is align with one of the slices, when performing the integration step-by-step on the view ray, we are by the way also advancing along this slice in the shadow map, from one light ray to the other, starting from $\gamma = 0$.

At each frame, to speed-up this process, Chen et al. compute a min-max mipmap tree, different for each slice. It takes the form of a binary tree, with as many leaves as light rays in the slice, which each contain the same angle β below which a view ray intersecting this light ray is shadowed. The role of the nodes is to merge the information regarding the minimum and maximum values of β contained in its two child nodes, and so on recursively, until all leaves are connected by a common root storing the minimum and maximum angles of the whole slice.

Instead of visiting each pixel of the slice and comparing the angles, using the min-max mipmap tree, the ray-marching is able to detect and skip entire segments of two, four, eight, or more values. At each step, knowing the current position γ in the slice, the tree is browsed from the root to leaf γ. On each node, if the view ray's angle is comprised between the minimum and maximum values

of the leaves, a comparison is still impossible, and we must continue to the lower level. If the view ray's angle is either below the minimum, or above the maximum, there is no need to visit the lower level, and all the leaves grouped by this node can be skipped, since we know that this portion of the view ray is completely lit or shadowed.

Conclusion This idea of taking epipolar geometry, mostly employed in researches where there is a need to establish correspondances between several images, and applying it to shadow mapping is very interesting. One of the advantages of this work by Chen and al. is its simplicity, since it seems that it can be easily plugged in an existing rendering pipeline. The use of min-max trees is also interesting for an implementation on the GPU, and can truely bring a substantial improvement from if we consider that in oudoor scenes, for example, we can expect large continuous sections of the view rays to be illuminated. However, like most illumination algorithms based on shadow mapping or involving a rendering of the scene from the light's point of view, because the screen is a plane and not a small point, the light cannot be positioned within the medium, and such effects as glows around the source cannot be rendered. Furthermore, where min-max trees may probably not fit a large number of participating media illumination algorithms is that most require performing some computations on the medium, even on shadowed portions, to take into account the heterogeneous optical depth along the view ray.

7.4 Conclusion

Fog rendering methods Fog, as a natural phenomenon, is the simplest and most widely used form of participating medium, which can easily be added to a wide variety of outdoor scenes. Contrarily to clouds or smoke, fog combines smoothness, a low density and, at the observer's scale, a large size.

Although fog can physically be caused by clouds floating at a low altitude, it would be much less disturbing to see outdoor fog having an illumination only approximated by a carefully chosen constant color. Because of its low density, light can penetrate fog very easily and reach the ground and other solid surfaces, on which it bounces. Due to the high albedo of fog and clouds, light can be scattered so much that it becomes impossible to identify the incident direction.

Cloud rendering methods We could see that clouds are one of the most challenging participating media to simulate and illuminate. Although there are many types of fog and smoke, faithfulness to the physical equations is not required as much as it is for clouds. Clouds are simulated in applications like flight simulators or meteorological simulations, which are very demanding in visual accuracy. We have seen [DKY+00, LHCL05, Dom06] that an acceptable cloud formation cycle can be rather inexpensively simulated using cellular automata.

Unlike fog, clouds are observed from outside, at a large distance, and well reflect light, which makes the directionality of their illumination very visible. In comparison with fog, one would naturally expect small (at the Earth's scale), dense and well bounded cumulus clouds to appear illuminated on their side facing the sun and much darker on the opposite side due to self-shadowing. This is particularly visible at sunset, when the inclination of sun rays approaches the horizontal. Additionally, we could see in methods like [Bou08] that, because of their high albedo, simulating multiple-scattering is much needed as it provides a real difference with single-scattering.

The majority of cloud rendering methods adopt the uniform voxels grid to model the medium. In [REK+04], we saw that atmospherical scattering phase functions cannot be ignored (Mie scattering, for clouds), since they are the key to obtain optical effects like glory. This is particularly adaptated to small and sparse volumes in a large scene, where a different grid can be used for each cloud. As in [NDN96], blobs can also be used to design the most common cloud types (cumulus, cumulonimbus, etc.), and are convenient to illuminate.

Smoke rendering methods In comparison with fog and clouds, smoke, generated from combustion, features a lower albedo. For this reason, it looks much darker, due to a greater absorption. The majority of smoke rendering works do not consider multiple-scattering (except a very local approximation [ZRL+08]), which does not provide the same improvement in visual realism as it does for clouds.

When looking at the literature, we can notice that smoke is, in a way, the most generic medium of the three. Apart from works specialized in its physical simulation like [FSJ01], pure visualization methods labelled as dedicated to smoke in particular are not the majority, since most of such methods can actually handle a wider range of heterogeneous media. Smoke can show a wide variety of shapes, this is why all works use true 3D models (except [RNGF03], where 2D models are stacked to approximate a 3D simulation), and it would make no sense to design homogeneous or analytical smoke. We saw in [ZRL+08] that, in contrast to old heterogeneous media methods, the advent of programmable GPUs makes it possible to use more sophisticated models that combine in the same algorithm both a representation using blobs and a representation based on a 3D grid. Particles like RBFs are incomparably useful to design a medium by hand as well as for simple non-physical animations. In [ZHG+07b], we see that some approximations can ease the evaluation of the optical depth between the sources and particles, making it possible to compute single-scattering without the use of a grid of samples at all.

Our verdict In the next chapter, we begin the presentation of our work, starting with a method to render outdoor fog in real-time. One of the key aspects is the heterogeneity of the medium, we therefore exclude full-analytical models like [LMAK00]. For the modelization, the use of function bases like in [BMA02] is very interesting, and enables a true heterogeneity of values in the grid of coefficients, while providing a smooth result by the product with basis functions with a small support. We were also inspired by [ZCS07], who renders horizontal layered fog using a couple of shaders on the GPU. More indirectly, we found interesting the idea by [Zdr04] of modeling the medium as a sum of layers of details with different resolutions, but exploited it using wavelets, instead of Perlin noise.

To model and render more indoor heterogeneous medium similar to smoke, we liked the idea in [ZRL+08] of combining both a regular grid of voxels and a set of RBFs. Regarding single-scattering illumination, in comparison with [MHLH05], who obtains beautiful results, we wanted both the lights and the medium to do not be fixed. The half-angle slicing method was interesting, before programmable GPUs emerged, as an elegant manner of avoiding useless and redundant optical depth integrations along similar paths. However, we could see in both [Kap09] and [Fat09] that distinct techniques based on iterative propagation of light in a grid could provide the same advantage, while avoiding the need to render the medium from the source, which is a problem if we aim at simulating effects like glows around lights inside the smoke. Even if we considered the

grid more adaptated to illumination using a two-step process like in [KVH84], we were convinced by [ZHG+07b] to not completely discard the use of RBFs since it can make design and animation of the smoke easier, in case that our rendering algorithm is not directly plugged to a third-party simulation which outputs voxels. The core of the algorithm was inspired by the Light Propagation Volumes [Kap09], where we found the idea of propagating radiance step-by-step inside a discrete volume flexible and promising, even if it had to be modified for direct lighting and, most of all, for handling participating media.

Part III

Real-time rendering of heterogeneous media using the GPU

Chapter 8

Real-time rendering of fog with dynamic low-frequency details removal

"The charlatan is always the pioneer. From the astrologer came the astronomer, from the alchemist the chemist, from the mesmerist the experimental psychologist. The quack of yesterday is the professor of tomorrow."

Sir Arthur Conan Doyle, *The Leather Funnel*

8.1 Introduction

8.1.1 Foreword

Fog is used in rendering both for aesthetic purposes and to increase performances by providing an efficient way to cull surfaces that are far from the camera. Simple fog models, are straightforward to implement but, like OpenGL's fog model, only allow a basic representation of homogeneous fog as can be seen on figure 8.1. Most of the time, these models are barely convincing visually, as we know that natural fogs never reach such perfect homogeneity. Considering latest advances in GPU programming, design of heterogeneous fog should be simple, and its rendering reachable in real-time.

The fog phenomenon is due to small particles of water in suspension. Because it interacts with light rays, fog is considered as a participating medium in computer graphics. Fog effects take into account attenuation, caused by absorption and out-scattering, and also consider multiple scattering of light as isotropic and constant over the scene. If we consider an homogeneous fog in its simplest form, equations are simple enough to allow an analytical integration of its effects along a view ray. When rendering heterogeneous fog, the density of water particles is varying across the scene, thus dramatically complexifying the model, involving local changes in physical properties of the fog, such as its extinction coefficient. Therefore, in order to compute the light-fog interaction,

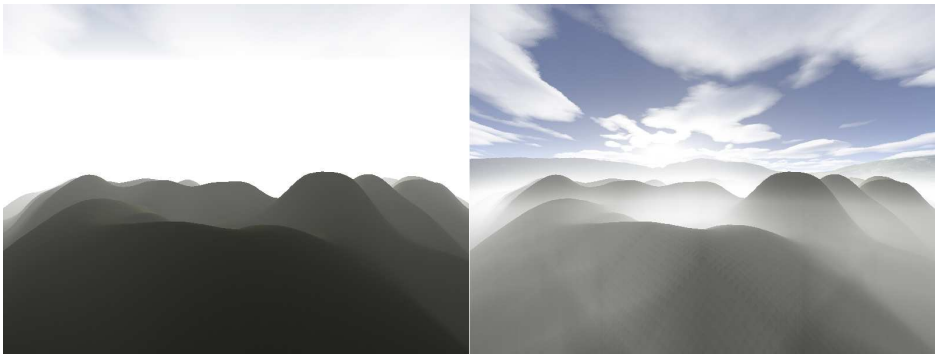

Figure 8.1: Comparison between opengl's homogeneous fog (left) and heterogeneous fog (right).

we have no other choice than performing the integration of the density along each view ray from the eye to the nearest object.

8.1.2 Motivation

The motivation behind this work originated from the constatation that, at the time (years 2008-2009), there existed no fog rendering method which allowed both to :

- Easily model smooth heterogeneous outdoor fog, in a way that allows a local control of the density distribution.
- Render the fog by taking advantage of the capabilities of programmable graphics hardware.

If we consider the first aspect, we can cite Biri et al. [BMA02], who were the firsts to design such fog. They advocate the use of function bases to accurately achieve a smooth density distribution, and use the functionnalities available on non-programmable GPUs, such as blending and texture mapping to accelerate the rendering.

Other related works, such as Zhou et al. [ZCS07], only briefly propose, to obtain heterogeneous fog as an extension of their layered homogeneous model, to trivially assign a different 1D function to each axis. Although the fog is rendered as a post-processing step using shaders, their strategy is far from offering a true local control of the fog's density, and for this reason remains very unsatisfying.

Some researchers considered the use of Perlin noise to design a true 3D fog, like Zdrojewska [Zdr04]. In this case, the user has some sort of control over the parameters of the noise, such as the manner in which the different octaves are combined, but has, by definition of noise, absolutely no local control over the fog's density. Moreover, although she uses the GPU with a couple of Cg shaders, Zdrojewska's fog cannot be considered as a true heterogeneous model, since no actual integration of the noise's density is performed. Instead, this integration is approximated by a homogeneous distribution integrated analytically between the camera and the nearest surface, which is then perturbated by the local noise values only fetched at the surface's position.

8.1.3 Quick overview

This the first method on which we have been working for this thesis. In this chapter, we present a new method helping to design and render complex heterogeneous fog in large outdoor scenes. In this work, the illumination by a single light source, the sun, is approximated by a constant term. However, a true single-scattering model with shadow effects is presented later in this manuscript (chapter 11).

Our method is organized in three different steps :

- First, the fog is modeled in a B-Spline function basis, which allows a simple and efficient construction of its extinction function. Because the density is modeled using smooth functions, we avoid common visual artifacts such as abrupt transitions, often occuring with a grid of voxels.

- Second, as a preparation for an optimized rendering, Mallat's wavelet decomposition is applied on the extinction function, from which different layers of low to higher frequency details are extracted.

- Finally, at runtime, the fog is rendered in real-time using GLSL shaders on the GPU, based on an accurate ray-marching on the density function from the viewer to the nearest surface. Because the different frequencies were separated from the density distribution, we are able to accelerate the visualization by selectively discarding the thinnest details in the fog which are not visible above a given distance from the viewer, while all frequencies are still rendered for areas near the camera.

In comparison with an extraction of the details based on a simple classical averaging and downsampling of the density map, using wavelets offers several advantages.

First, like all functions basis, it combines the advantages of a discrete representation, enabling sharp transitions in neighbouring coefficients, while having the insurance of a smooth continuous result, because of the multiplication of such coefficients by smooth basis functions.

Then, the number of values modeling the fog is kept almost unchanged after the extraction of the pyramid of frequencies as in the original one-layered function basis. As a result, apart from the interesting gain in memory especially on the GPU, we are certain that, in the worst situation where all frequencies are visible from the viewer, in comparison with a ray-marching on the original density function, by performing successive ray-marchings all way long through each extracted layer of details, we do not integrate over more coefficients.

Finally, a simple downsampling of the fog's density only provides an accurate result on a grid of samples, but leads to an inexact result with a function basis, when the values of the grid have to then be multiplied by a basis function. Instead, the wavelet decomposition algorithm, specifically adaptated to each type of wavelet, accurately separate the approximation and the associated details from the input function basis without any loss or alteration.

Therefore the contribution of this work is :

- Introducing a method combining an easy modelization of a truely heterogeneous fog, the advantages of smooth functions bases, and exploiting the capacities of programmable GPUs.

- Ours is the first method using wavelets and scaling functions to model a participating medium.

- We present en efficient strategy to speed-up the rendering by exploiting wavelet compression to dynamically discard the less contributing details.

8.2 Our fog model

Between points O and P, the fog induces an attenuation (due to out-scattering and absorption) of the luminance L of P and an increase (in-scattering and emission) of light along the ray $\vec{OP}$.

We begin by considering the integral transfer equation, see [SH92] :

$$L(O) = \tau(O,P)\ L(P) + \int_{u=0}^{|OP|} \tau(O,x(u))\ \rho(x(u))\ \beta(x(u))\ F(\alpha)\ J(x(u))\ du, \tag{8.1}$$

where $L(O)$ is the radiance received by the observer, $x(u)$ is the position on the view ray which corresponds to the parameter u, $J(x(u))$ is the incoming radiance along the ray, $\rho(x(u))$ is the local density at position $x(u)$, $\beta(x(u))$ is the local absorption coefficient at $x(u)$, F is the phase function, α the scattering angle, and $\tau(x(u),v)$ the optical depth of the fog along the ray going from u to v :

$$\tau(x(u),v) = \mathrm{e}^{-\int_u^v \rho(x(s))ds} \tag{8.2}$$

Our main goal is to render our fog in real-time, using conventional graphics cards. Although performances of GPUs have never been increasing so fast, we have to apply several simplifications on our fog model.

In this work, we make the choice to do not illuminate the medium using single or multiple-scattering. Instead, like in [BMA02], we assume that because the medium will always be inserted in an outdoor scene, the incident sunlight will scattered within the fog to the point that it becomes impossible to know where it comes from. In this situation, the radiance $J(x(u))$ arriving at $x(u)$ before being scattered towards the viewer can be approximated by a constant amount L_{fog}.

Equation (8.1) then becomes :

$$L(O) \simeq \tau(O,P)\ L(P) + \int_{u=0}^{|OP|} \tau(O,x(u))\ \rho(x(u))\ \beta(x(u))F(\alpha)L_{\text{fog}}\ du \tag{8.3}$$

Finally, we assume isotropic scattering and constant absorption β, such that :

$$\beta(x(u))\ F(\alpha) = \beta \tag{8.4}$$

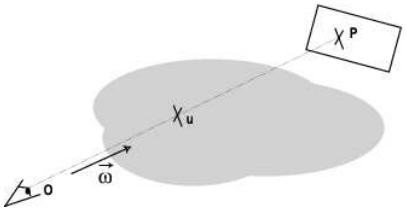

Figure 8.2: Ray of incoming light from P to O through a participating medium.

We now have :

$$L(O) \simeq \tau(O,P)\ L(P) + \beta \int_{u=0}^{|OP|} \tau(O,x(u))\ \rho(x(u)) L_{\text{fog}}\ du \tag{8.5}$$

The second part of equation (8.5) can be reformulated for an analytic integration. We finally obtain, as our final model :

$$L(O) \simeq \tau(O,P)\ L(P) + \beta L_{\text{fog}}\ (1 - \tau(O,P)) \tag{8.6}$$

8.3 Our method

8.3.1 Modeling the fog

8.3.1.1 A layered two-dimensional heterogeneous medium

Unlike other types of participating media such as clouds or smoke, fog almost always appears in large outdoor scenes as a horizontal layers of varying thickness. This is quite different from smoke, which can evolve indifferently in all directions in terms of shape and movement, and thus really needs to be defined with the same precision along all three dimensions. For this reason, and in order to ease the design step as much as possible and, later, the visualization using the GPU, we choose to restrict its true heterogeneous representation in two dimensions.

The optical properties of our fog, similarly to most other participating media rendering techniques, are proportional to its density, which itself depends on its extinction function. The fog's density is modeled using a 2D B-Spline function basis, and is therefore fully heterogeneous at least along the two X and Z dimensions. Vertically, we choose to use the same principles as layered fogs [ZCS07], where the density decreases linearly or exponentially when the difference between the altitude of the point evaluated and that of the fog increases.

In a nutshell, the complete density function ρ can be expressed as the following product :

$$\rho(x(u)) = \rho^{XZ}(x(u))\ \rho^{Y}(x(u)), \tag{8.7}$$

where $x(u)$ is a position along the view ray, ρ^{XZ} is the horizontal non-analytical component of the density distribution, modeled in a B-Spline functions basis, and $\rho^{Y}(x(u))$ is the vertical layered extinction term, defined as :

$$\rho^{Y}(x(u)) = \gamma^{|\text{fog}_y - x(u)_y|}, \tag{8.8}$$

where γ is the vertical extinction coefficient, between 0 and 1, acting as a parameter defining how fast the density decreases vertically, and fog_y is the altitude of the fog.

8.3.1.2 Designing the fog's shape

The horizontal variations of our fog's density are modeled by specifying the value of each coefficient in the extinction function basis. These coefficients can be adjusted by hand, or be, for example, the result of a simulation, which is then exported as what we call a *fogmap* (see figure 8.9), i.e.

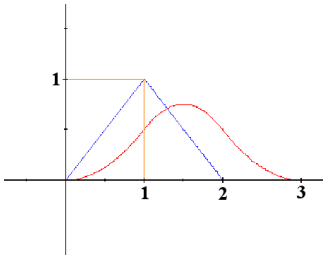

Figure 8.3: Shape of Haar, Linear and Quadratic B-Splines.

a simple greyscale image similar to a heightmap but where the pixel values represent densities instead of different altitudes, and then loaded in our implementation.

Compared with other techniques such as RBF or particle-based methods, shaping our fog using a grayscale image is quite straightforward. The fogmap represents, in some extent, a direct preview of its aspect, which can be interesting for some applications where great intuitivity is needed. To ease the manual setting of the coefficients, we also developed a small application where the values of the density can be directly adjusted using a drag-and-drop interface.

8.3.1.3 Choosing the type of functions

The desired appearance of the fog is the key criteria to consider when choosing the type of function used to model the density. It is clear that abrupt changes in density would not look natural, so we would ideally like continuous functions to design smooth fogs using as few coefficients as possible. In order to avoid border effects, the scaling function must tend to zero on both sides of its support, which eliminates, for example, Legendre scaling functions.

For design and optimisation purposes, our rendering algorithm also needs the scaling function never to oscillate under zero. Whatever the trajectory of the ray within the function in 2D, and more generally within the fog, we would like to be sure that the sum of the density it intersects can only increase as it traverses the fog from the observer to the nearest object. Daubechies wavelets, which, by the way, are not symmetrical, might not be the way to go.

Finally, we have to consider the fact that, as will be discussed in the next section, the cost of using a particular type of wavelet is quadratically proportional to the support of the scaling function in one dimension.

According to their shape, the most adapted candidates seem the linear or quadratic B-Splines, which are shaped like a Gaussian (see figure 8.3), and have a relatively compact support.

Although we are limited to wavelet scaling functions for the fog's representation, our method is not reduced to a particular type of wavelet. Our implementation specifically handles all degrees of B-Spline wavelets, but can be extended to other families, as long as they are compatible with Mallat's decomposition.

8.3.2 Preparing data for rendering

At this step, the type of basis function to use for modeling the horizontal component ρ^{XZ} of the medium's density has been choosen. These functions decide the visual smoothness of the fog and, as will be discussed, the complexity of the rendering step. A value was also assigned to each function basis coefficient, which comes as a factor to each of these functions. We can now consider

the first step of our rendering algorithm : the precomputations, where a wavelet transform is first applied on the *fogmap*, before its output is packed into several textures then transmitted to the GPU.

8.3.2.1 Decomposition of the *fogmap* ρ^{XZ}

These precomputations are crucial for our method. Although we initially only modelized the medium's density distribution as a single layered two-dimensional function, we need to be able to distinguish, at rendering, the details that matter most visually, from those that contribute less. An unavoidable criteria for deciding whether some details should be discarded or not is their period (or "size"), since whatever the situation, the largests details are always more visible than the smallest ones.

Because the density distribution was modeled in a B-Spline basis, we are now able to appeal to B-Spline wavelets and take advantage of multiresolution analysis. We apply N steps of Mallat's fast wavelet transform in two dimensions on the *fogmap*, and transform our single-layered density distribution into a first set of B-Spline basis coefficients. These coefficients represent a coarse approximation of the density at resolution $1/n$, plus N different sets of B-Spline wavelets coefficients in which the high-frequency details removed missing to the approximation are now scattered, according to the minimum resolution at which they appear.

This is an advantage of wavelets, that we do not have significatively[1] more coefficients after the decomposition than in the original signal. In comparison, a more naive multiresolution approach based on a simple downsampling of ρ^{XZ} would lead to, first, at least the same quantity of coefficients to represent the thinnest layer of details, plus all the coefficients needed for the approximation and all other layers of details.

8.3.2.2 Different types of textures

The modeling of the fog and the wavelet decomposition are both performed in the application, on the CPU. Since the rendering algorithm is implemented on the GPU, we therefore need to transmit a certain number of elements from the CPU to the GPU. Some of these precomputations are transmitted as two-dimensional textures.

Textures storing all bases coefficients

The first type of texture contains the different sets of coefficients generated by the N steps of the decomposition. Because the number of texture units available in OpenGL is limited, we chose to pack together in one single texture all layers of details using the same type of basis function. We therefore have to transmit a total of four separate textures of coefficients. The approximation texture will always only contain one single image, whereas each of the three textures associated to each of the three types of details contains N sub-images disposed horizontally from the lowest to the highest frequency. Because the sub-images have different dimensions, in order to ease the access to the values from the shader, they are separated by a constant distance corresponding to the size of the largest layer of detail.

[1] It depends on the type of wavelets. With Haar wavelets, no additional coefficients are generated by the decomposition. With more sophisticated functions like linear or quadratic B-Splines, we respectively have one or two lines and columns added per layer of details.

Precomputing sampled integrals across the basis functions

We also generate a second set of images, that we call *integral patches*. In these images, we precompute a series of integrals through the definition domain of each basis function $\phi\phi$, $\phi\psi$, $\psi\phi$ and $\psi\psi$. This second set of precomputations can be justified by, first, the impossibility to reach real-time if the integration had to be evaluated at runtime for coefficient cell traversed by the view ray and, second, the fact that all coefficients of all layers of the same type of details or approximation that are traversed by our ray-marching need to be multiplied by an integral over the same basis function.

To understand how the integrals are precomputed and organized in these patches, one has to bear in mind that, depending on the degree of B-Spline, the support (size of the definition domain) of both the 1D scaling function ϕ and the 1D wavelet ψ may vary, in the sense that our 2D functions $\phi\phi$, $\phi\psi$, $\psi\phi$ and $\psi\psi$ can have a square or rectangular support. The 2D rectangular support of each function is cut in an series of 1×1 square intervals on which the integrals are precomputed separately, and which form as many distinct patches. We sample $n \times n$ regularly spaced positions within this 1×1 interval, and then evaluate, across this 1×1 square, all n^4 possible integrals between two sampled positions. Inside a patch, the integral between sampled positions (a, b) and (c, d) is stored at pixel $(a \cdot b, c \cdot d)$.

If the basis function ϕ has a support of s_1 and ψ a support of s_2, this means that we must generate a total of $s_1^2 + s_1 \cdot s_2 + s_2 \cdot s_1 + s_2^2$ patches. All these patches can then be assembled in a single 2D texture (dimensions : $[(s_1 + s_2) \cdot n^2] \times [(s_1 + s_2) \cdot n^2]$), and transmitted to the GPU.

8.3.3 Rendering the fog

At this step, the fog has been modeled in a 2D B-Spline function basis, provided in input to our pipeline. As a first precomputation, this B-Spline function basis was decomposed in an approximation and three different pyramids of details of different frequencies, respectively in a B-Spline basis and in three sets of wavelets bases. This data was transposed into a first type of textures, while a second set of textures was created in which are stored a number of integrals along a ray traversing each basis function between two sampled positions on the border of its definition domain.

8.3.3.1 Overview

The goal of this last step is the final visualization of the whole foggy scene, composed of classical solid surfacic geometry, provided to our implementation by a simple pointer towards a function directly drawing type of scene, and the inserted participating medium, whose presence between the camera and the surfaces is expected to alter their color. To achieve this, we first need to compute the visual aspect of the fog alone, and then blend the result with the color $L(P)$ of the nearest surface to obtain the final pixel color $L(O)$.

For each pixel, to compute the appearance of the medium, we perform a ray-marching from the camera to the nearest object, during which we integrate over the fog's extinction function ρ to obtain the total optical depth $\tau(O, P)$ along the view ray $\vec{OP}$. This optical depth is then involved in the blending between the color of the object and that of the fog, as shown in equation 8.6. We implemented this ray-marching on the GPU, using GLSL in a vertex/fragment shaders couple. Algorithm 9 presents a pseudo code version of our fragment shader, for the simple case where basis functions do not overlap each other.

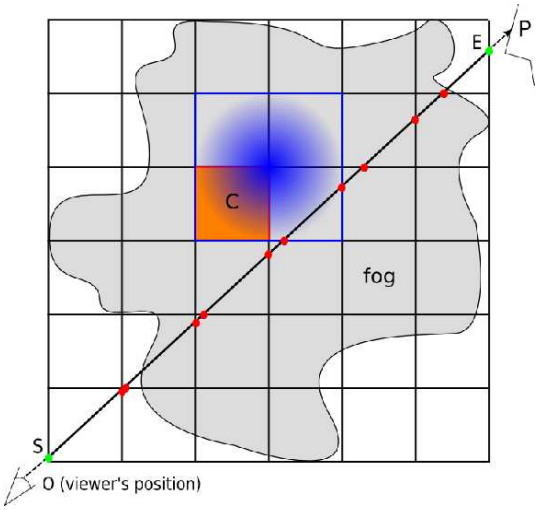

Figure 8.4: Ray-marching through a single level, designed with linear B-Splines scaling functions (support=2).

8.3.3.2 The grid

As a result from the wavelet decomposition, the fog's density is scattered in several multiple-level function bases, each having their own vector space and definition domain in 2D. Each single level can be assimilated to a rectangular grid, each cell being associated to both a coefficient and a basis function. Since all bases have the same definition domain, the grids from different bases match for each given level.

8.3.3.3 Integration along the ray

Overview

We start by transposing both positions of the camera and the object from the scene to the fog's vector space. This task is performed when entering the fragment shader by a simple product of both positions with a matrix precomputed on the CPU.

Our algorithm performs the entire integration level by level, and then, for each single function basis level, cell by cell.

To initiate the integration on a given level, we first determine both entry and exit points of our integration on the grid. The entry point corresponds to either the nearest intersection between the ray and the current level's bounding box, or the viewer's position in case he stands within the fog. Similarly, the exit point corresponds to the intersection with either the farthest plane of the bounding box, or with the nearest object if situated within the fog.

The algorithm is iterative, but instead of advancing regularly along the ray to then naively sum local density values, we move cell by cell, fetch and accumulate our precomputed integral values. At each ray-marching step, we find ourselves lying between two positions situated on the perimeter of one cell : the entry $x(t_i)$ and exit $x(t_{i+1})$ intersections with the view-ray.

To compute $\tau(O, P)$, we need to accumulate all small integrals on ρ over the portion of each cell traversed by the view-ray :

$$\tau(O,P) = \exp\left[-\sum_i \left(\int_{u=t_i}^{t_{i+1}} \rho(x(u))\ \mathrm{d}u\right)\right] \tag{8.9}$$

From (8.7), we can expand (8.9) :

$$\tau(O,P) = \exp\left[-\sum_i \left(\int_{u=t_i}^{t_{i+1}} \rho^{XZ}(x(u))\ \rho^{Y}(x(u))\ \mathrm{d}u\right)\right] \tag{8.10}$$

Approximating the product $\rho^{XZ}(x(u)) \cdot \rho^{Y}(x(u))$ on each cell

The fog's density ρ is obtained as the product of two distinct components, the horizontal set of function bases $\rho^{XZ}(x(u))$ and the vertical analytical attenuation $\rho^{Y}(x(u))$. We know that, at the mathematical level, the integral of a product between two functions f and g is not equivalent to the product $\int f \cdot \int g$ of their separate integrals.

Unfortunately, we are nevertheless compelled to use this solution :

$$\int_{u=t_i}^{t_{i+1}} \rho^{XZ}(x(u))\ \rho^{Y}(x(u))\ \mathrm{d}u \simeq \int_{u=t_i}^{t_{i+1}} \rho^{XZ}(x(u))\ \mathrm{d}u \cdot \int_{u=t_i}^{t_{i+1}} \rho^{Y}(x(u))\ \mathrm{d}u \tag{8.11}$$

Because the two functions ρ^{XZ} and ρ^{Y} are evaluated with the same position $x(u)$ on the view ray, $\rho^{XZ}(x(u)) \cdot \rho^{Y}(x(u))$ is not a product between two independant factors. In this case, we can hardly find a better approximation than equation 8.11, without actually performing a thinner numerical integration on $\rho^{XZ} \cdot \rho^{Y}$ within the cell. Even if we have an analytical expression for ρ^{Y}, which is given by equation 8.8, we lack an analytical expression for ρ^{XZ}, whose integral over a cell is precomputed and transmitted in a texture. Our choice is therefore to integrate over $\rho^{XZ}(x(u)) \cdot \rho^{Y}(x(u))$ as accurately as we can, indeed using the somewhat coarse precision offered by the length of each iteration in the grid traversal.

Integrating over the fogmap ρ^{XZ}

From (4.38), we know that in order to evaluate the density at a given position, we must evaluate it at the approximation and all layers of all three types of details, and sum the results :

$$\rho^{XZ}(x(u)) = A^{\phi\phi}(x(u)) + D^{\phi\psi}(x(u)) + D^{\psi\phi}(x(u)) + D^{\psi\psi}(x(u)), \tag{8.12}$$

where $A^{\phi\phi}$ is the approximation function basis, and $D^{\phi\psi}$, $D^{\psi\phi}$, $D^{\psi\psi}$ represent the complete pyramids of all layers of each of the three types of details.

If we express (8.12) as a sum over the different levels, we have :

$$\rho^{XZ}(x(u)) = A_0^{\phi\phi}(x(u)) + \sum_{j=0}^{N-1} \left[D_j^{\phi\psi}(x(u)) + D_j^{\psi\phi}(x(u)) + D_j^{\psi\psi}(x(u))\right], \tag{8.13}$$

where $D_j^{\phi\psi}$, $D_j^{\psi\phi}$, $D_j^{\psi\psi}$ represent the particular layer of level j in each multiresolution pyramid, and $A_0^{\phi\phi}(x(u))$ is the unique layer of the approximation.

At rendering, we compute $\int_{u=0}^{|OP|} \rho^{XZ}(x(u))\ \mathrm{d}u$ by successively integrating over each single layer, this can be written :

$$\int_{u=0}^{|OP|} \rho^{XZ}(x(u))\ \mathrm{d}u = \int_{u=0}^{|OP|} A_0^{\phi\phi}(x(u))\ \mathrm{d}u + \sum_{j=0}^{N-1} \left[\int_{u=0}^{|OP|} D_j^{\phi\psi}(x(u))\ \mathrm{d}u + \int_{u=0}^{|OP|} D_j^{\psi\phi}(x(u))\ \mathrm{d}u + \int_{u=0}^{|OP|} D_j^{\psi\psi}(x(u))\ \mathrm{d}u \right] \tag{8.14}$$

To explain how the integration over one single layer is performed, we take the particular example of level j in the pyramid of details of type $\phi\psi$. We iteratively advance cell-by-cell. On the i^{th} intersection between the view-ray and a cell, we compute the small integral corresponding to this segment, accumulate the result and move on to the next cell, intersected between t_{i+1} and t_{i+2} on the ray :

$$\int_{u=0}^{|OP|} D_j^{\phi\psi}(x(u))\ \mathrm{d}u = \sum_i \int_{u=t_i}^{t_{i+1}} D_j^{\phi\psi}(x(u))\ \mathrm{d}u \tag{8.15}$$

We stop the ray-marching when parameter t_i on the view-ray corresponds to a position outside the fog's bounding box.

Integrating ρ^{XZ} on each cell

Integrating over a function basis is much more trickier than over a simple function. As shown by equation (4.1), if we evaluate the function basis at a position x naively, we actually have to evaluate the value at position x on every small basis function. For the general case, this is justified, since several basis functions may overlap on position x, and therefore contribute to the final value.

When computing the contribution of one particular cell, we find ourselves exactly in the same situation. When using Haar wavelets and scaling functions, which are defined on an 1×1 support, the algorithm is much simpler, since neighbouring basis functions do not overlap on other cells than the cell containing their own coefficient. With other types of functions, for instance the quadratic B-Spline wavelets with their support of 5×5, we must take into account the function attached to the current cell, as well as 24 other neighbouring functions whose definition domain also overlap on this cell.

With non-overlapping basis functions (Haar) We first place ourselves in the simplest case where the support of each basis function $\phi\phi_{j,p,q}$, $\phi\psi_{j,p,q}$, $\psi\phi_{j,p,q}$ or $\psi\psi_{j,p,q}$ matches the area of its own cell $V_{j,p,q}$ on the grid. p and q are the integer indices in the grid of the cell intersected between $x(t_i)$ and $x(t_{i+1})$ along the view-ray. To compute the integral of one function basis over one cell, we therefore only have to take one small basis function into account.

In this situation, if we expand (8.15) to make appear the product between the basis coefficient $d_{j,p,q}^{\phi\psi}$ and the basis function $\phi\psi_{j,p,q}$, we have :

$$\begin{cases} \displaystyle\int_{u=0}^{|OP|} D_j^{\phi\psi}(x(u))\ \mathrm{d}u = \sum_i \int_{u=t_i}^{t_{i+1}} d_{j,p,q}^{\phi\psi}\ \phi\psi_{j,p,q}(x(u))\ \mathrm{d}u \\ \qquad \text{with } \{p,q\} \text{ such that } V_{j,p,q} \cap [x(t_i)\ x(t_{i+1})] \neq \emptyset \end{cases} \tag{8.16}$$

In our implementation, p and q are maintained and updated at each intersection depending on by which side we enter each new traversed cell.

We continue with equation (8.16) and simply extract the basis coefficient and place it outside the integral.

We have :

$$\begin{cases} \displaystyle\int_{u=0}^{|OP|} D_j^{\phi\psi}(x(u))\ \mathrm{d}u = \sum_i \left[\underbrace{d_{j,p,q}^{\phi\psi}}_{\mathbf{a}} \underbrace{\displaystyle\int_{u=t_i}^{t_{i+1}} \phi\psi_{j,p,q}(x(u))\ \mathrm{d}u}_{\mathbf{b}} \right] \\ \qquad\qquad \text{with } \{p,q\} \text{ such that } V_{j,p,q} \cap [x(t_i)\ x(t_{i+1})] \neq \emptyset \end{cases} \tag{8.17}$$

Written under this form, (8.17) expresses very explicitely how we implemented the computation of $\int_{u=0}^{|OP|} D_j^{\phi\psi}(x(u))\ \mathrm{d}u$.

Where the algorithm is actually much less complicated than it can seem at first sight, is because all basis coefficients and integrals have been precomputed and ready to be fetched from the textures. In equation (8.17), the highlighted value a is read from the appropriate texture of coefficients, and the integral b is simply extracted from our precomputed patches.

Figure 8.4 shows ray $\vec{OP}$ traversing a single level's grid from entry point S to exit point E. Integration steps (i.e. intersections with the grid) are shown in red. The basis function (in this example : linear B-Spline scaling function) associated to the orange cell's coefficient c is shown in blue.

With basis functions having a support larger than 1×1 When the functions are supported on a domain larger than 1×1, part of the contribution of each cell gets superimposed on that of its neighbouring cells, thus also contributing to rays which do not necessarily pass through those cells themselves. Actually, a ray passing through a cell must take into account the contribution of that cell, plus the contributions of the $dx - 1$ previous cells on the X axis, times the $dy - 1$ previous cells on the Y axis, where dx and dy are the dimensions of the basis function's definition domain.

We have :

$$\begin{cases} \displaystyle\int_{u=0}^{|OP|} D_j^{\phi\psi}(x(u))\ \mathrm{d}u = \sum_i \sum_{r=0}^{dx-1} \sum_{s=0}^{dy-1} \left[d_{j,p-r,q-s}^{\phi\psi} \int_{u=t_i}^{t_{i+1}} \phi\psi_{j,p-r,q-s}(x(u))\ \mathrm{d}u \right] \\ \qquad\qquad \text{with } \{p,q\} \text{ such that } V_{j,p,q} \cap [x(t_i)\ x(t_{i+1})] \neq \emptyset \end{cases} \tag{8.18}$$

8.3.4 Optimization & multiresolution

We exploit the main idea behind the wavelet transform, which consists in factorizing data as much as possible locally, so that coefficients are affected on layers which are as high as possible (i.e. corresponding to the coarsest resolutions), thus setting to zero as much coefficients as possible on the lower layers (i.e. corresponding to the highest resolutions), minimizing their contribution to the final density function.

Algorithm 9 Pseudo code of the shader

```
for each pixel do
  sum = 0
  for l = 0 to nb_levels do
    compute 2D entry point on grid
    compute 2D exit point on grid
    while pos ≠ exit do
      inter = compute next intersection with grid
      if (l = 0) then
        coef = get cell coef on approx basis
        approx = integrate on φφ between pos & inter
        sum += coef*approx
      end if
      coef = get cell coef on details1 basis
      det1 = integrate on φψ between pos & inter
      sum += coef*det1
      coef = get cell coef on details2 basis
      det2 = integrate on ψφ between pos & inter
      sum += coef*det2
      coef = get cell coef on details3 basis
      det3 = integrate on ψψ between pos & inter
      sum += coef*det3
      pos = inter
    end while
  end for
  optical_depth = exp(-sum)
  pixel_color=optical_depth*obj_color + (1-optical_depth)*absorption*fog_color
end for
```

Our optimization consists in omitting an increasing quantity of details from layers whom resolution is above a threshold which decreases as the observer moves away from the fog. When integrating the fog's density from the observer O to point P, the maximum integration distance $d_{\max_l}$ on level $l \in \mathbb{N}$ is given by :

$$d_{\max_l} = d_{\text{ref}} \times \mu^l \tag{8.19}$$

where $\mu \in [0.0, 1.0]$ is the optimization coefficient, and d_{ref} is a reference distance, above which layers of details from levels $l \geq 1$ are discarded. In our implementation, the firsts layers of all pyramids are not affected, and will always be rendered, so that even from a large distance, at least a very coarse fog is still visible. The reference distance d_{ref} is defined by the user, for example as a function of the size of the scene, or based on the parameters of the projection matrix and the resolution of the screen. When $\mu = 1.0$, the integration is performed almost entirely on all levels ; on the contrary, when $\mu = 0.0$, only the coarse approximation and the coarsest layers of details are rendered.

As seen previously, when using scaling functions that are defined on more than an 1×1 square,

the integration cost is no longer proportional to the fog's size, since more than each single particular cell traversed by the ray brings a contribution on these cell's area. That's why although the total number of coefficients modeling the fog stays almost unchanged, the rendering cost increases dramatically after the wavelet decomposition, since B-Spline wavelets always have a larger support than their scaling function.

When using such basis functions, for example linear or quadratic B-Splines, it can be interesting to use the two-scale relation for wavelets (4.7) to *deconstruct* the three wavelet bases. This turns them back into scaling function bases, which can then be merged (i.e. added) together. When using scaling functions with a large support, this operation, performed on the CPU just after the decomposition, can accelerate the rendering by a factor of two, while keeping the multi-resolution aspect brought by the decomposition. Moreover, if we perform a deconstruction, we can stop the integration as soon as the sum reaches a particular threshold, close to a great opacity. Deconstruction is important since it makes sure that each new cell will only add opacity.

8.4 Results and discussion

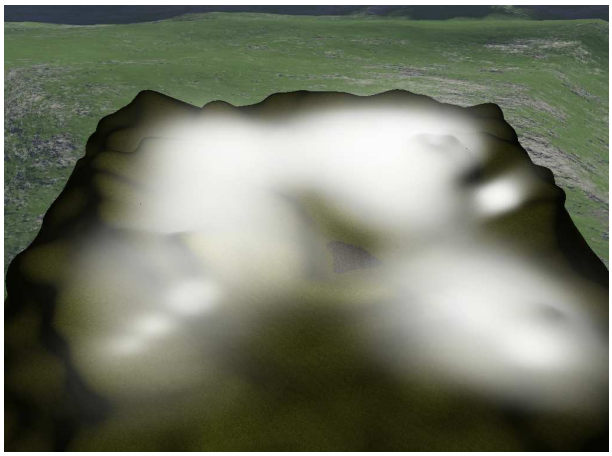

Figure 8.5: A very smooth fog modeled using quadratic B-Splines.

This algorithm has been implemented using GLSL, an Intel Core 2 Quad 2.8Ghz processor and a NVidia GeForce GTX 570 graphics card. Screen resolution is 1024×768.

8.4.1 Performance

Table 8.1 show FPS results obtained when using our ray-marching alone to directly render raw Haar, linear or quadratic fogmaps, without any decomposition. We clearly stay real-time, even with large fogmaps such as 64×64. We could almost handle higher resolutions, however our wavelet decomposition algorithm, which needs further optimizations, takes several minutes to decompose fogmaps larger than 80×80. Each type of basis function is defined on an area which size is increasing linearly in 1D, which involves a quadratically increasing number of neighbouring cells contributing to the density on each 1×1 square on the grid.

Table 8.2 show FPS results obtained when rendering a 64×64 Haar fog using our details dropping optimization, for different values of the tolerance parameter μ. Based on the size of the

scene, we chose $d_{\text{ref}} = 12.0$ for the reference distance, meaning that all details on levels $l \geq 1$ farther than a distance of 12.0 in the scene are discarded when $\mu = 1.0$, and more details are dropped as μ decreases. When we decrease μ, we go less far in our grid traversal, and stop it before the next surface is reached. The distance at which we stop integrating the density is different depending on the importance (i.e. the frequency) of the details, to favor low frequencies. When $\mu = 0$, we only render the coarse approximation and level 0 in all three types of details. If, in addition, we do not apply any decomposition step, we are directly rendering the fogmap, like in table 8.1. We can notice that, in our implementation, simply loading and rendering a fogmap (situation s_1) is actually more expensive than rendering the four functions bases resulting from one first step of wavelet decomposition (situation s_2). This can be explained by the fact that, on the one hand, the same number of coefficients is rendered in these two situations, and, on the other hand, in s_2 we process simultaneously within the same loop one cell from each pyramid. It results that s_1 involves traversing a twice larger grid than in s_2, in which several grids are processed together, and the high cost of the grid traversal makes the unprocessed original fogmap more expensive to render alone directly than after a first decomposition step.

However, this is not true anymore starting from two decomposition steps, where each new step generates new layers of details. When $\mu = 1$, each additional layer makes the whole rendering process more expensive, but presents the advantage that, when μ decreases, details are removed more rapidly, which explains the gain in speed.

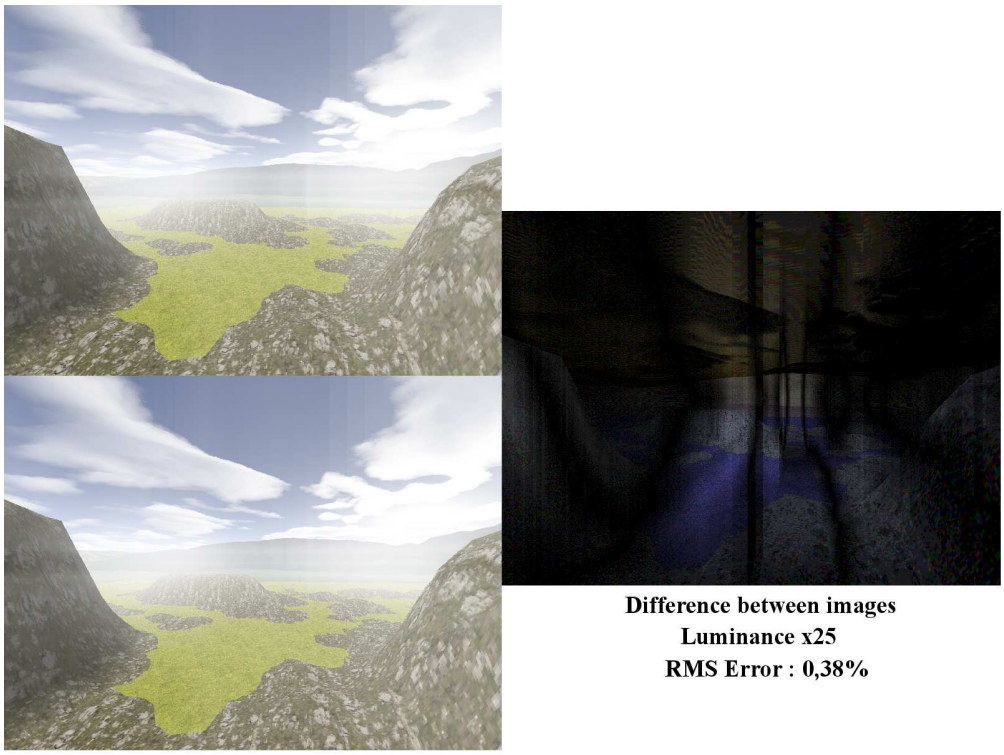

Figure 8.6: Quality difference when removing all layers of details (bottom left) from an Haar 32x32 (top left). Difference image (right) is shown (x25).

Fog resolution	Haar	Linear	Quadratic
16×16	564	427	304
32×32	386	267	170
64×64	235	153	90

Table 8.1: FPS results with our ray-marching without optimizations.

Nb steps \ μ	1	0.8	0.6	0.4	0.2	0
0	235	235	235	235	235	235
1	302	302	302	302	302	302
2	232	248	270	302	351	408
3	182	188	222	288	391	501
RMS error	-	0.078%	0.884%	2.445%	3.563%	3.763%

Table 8.2: FPS results with our optimization, using a 64×64 Haar fogmap, and a varying number of wavelet decomposition steps. The last row shows the error between the images obtained for $\mu < 1.0$ and $\mu = 1.0$ using 3 decomposition steps.

8.4.2 Visual quality

The higher the degree of the B-Spline wavelet is, the smoother each basis function looks. With Haar wavelets, we can see, in figure 8.7.B, that the visual result is a bit unsatisfactory, with abrupt changes in density which betray the discontinuity of Haar functions. With linear B-Spline wavelets (figure 8.7.C) the framerate decreases but the visual result is a lot smoother and artifacts and peaks are now practically imperceptible. Finally, with quadratic B-Spline wavelets (figure 8.7.D), we loose in performance but this time, the quality gain is relatively low compared to linear B-Spline wavelets.

8.4.3 Discussion

Linear B-Spline seems a good trade-off between speed and quality but Haar could be used if rendering time is an issue. The advantage of using wavelets, beside their property of good data compression, is to have a mathematical representation of heterogeneous fog from physical simulation to rendering. Indeed, animating such fogs is easy, since wavelet decomposition can be performed in real-time. Moreover, unless previous approaches, our integration of the density along the view ray really computes the intersections with the cells traversed in the grid, instead of using a fixed-step ray-marching. Based on our precomputed integral patches on pattern cells, we obtain results close to an analytical integration. In comparison to particle approaches like [ZHG+07b], our method is more adapted to large outdoor scenes when camera is moving inside the fog, and the modelling is far more intuitive than using particles.

One important limitation of our method is related to the size of the fogmap that can be loaded in our application. While the rendering step on the GPU is not a problem, our implementation of the wavelet decomposition starts taking more than a minute to process a 80×80 or larger fogmap, especially using B-Spline wavelets. It could be interesting to parallelize this process using CUDA,

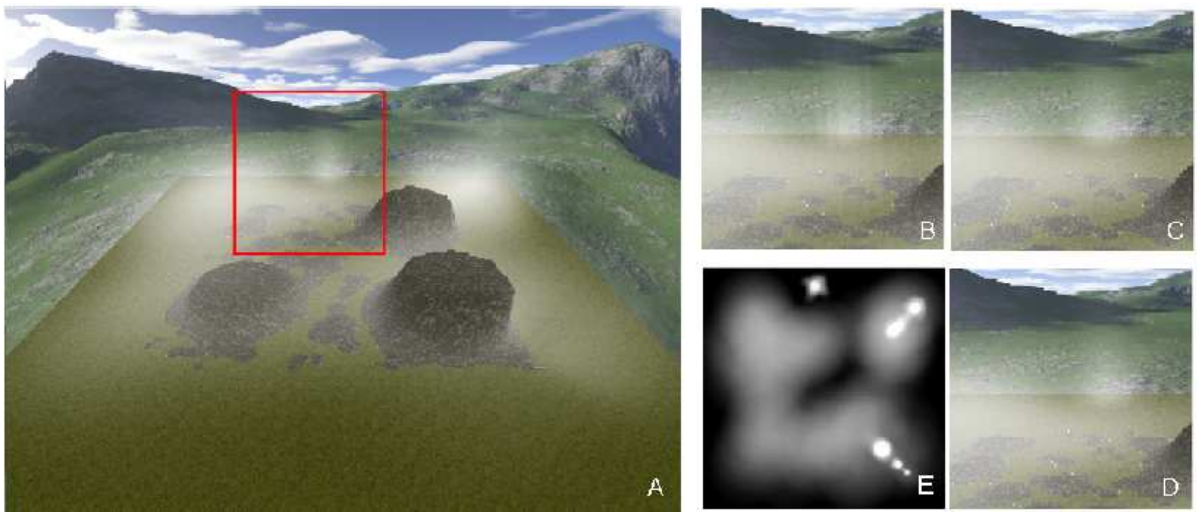

Figure 8.7: Quality difference with a large 30x30 fog. Fog taken from above (A), and the associated fogmap (E). Zoom on the red part when using Haar (B), Linear (C) and Quadratic (D) wavelets.

for example.

8.5 Conclusion

In this chapter, we presented a new method for modeling heterogeneous fog using wavelet scaling functions. Rendering is performed through a simple decomposition scheme of the fog density function represented in a scaling function basis leading to sparse data. Wavelets and scaling functions allow and ease a certain number of precomputations, such as the integrals of the wavelets along each ray. A brute force rendering algorithm using the GPU has been presented allowing real-time rendering for moderated complex fog along with an optimized version taking profit of the sparcity of data induced by the wavelet decomposition. Our method does not depend on the position of either the sun or the fog, allowing simple transformations of the fog.

The use of wavelets opens the door to other major optimisations for our method. Mainly, the rendering algorithm can be improved by focusing only on the grid's cells which actually contain a non-negligible value, in order to be able to directly jump to the interesting zones of the fog when performing the integration along the ray. For this purpose, we aim at designing a simple GPU traversal of the graph generated by the wavelet decomposition. Since wavelets can be used to solve fluids equations, we also plan to link our rendering algorithm to a physical simulation involving wavelets, allowing a real-time physical animation and rendering of heterogeneous fog. Finally, we plan to add single scattering and volumetric shadows in our model.

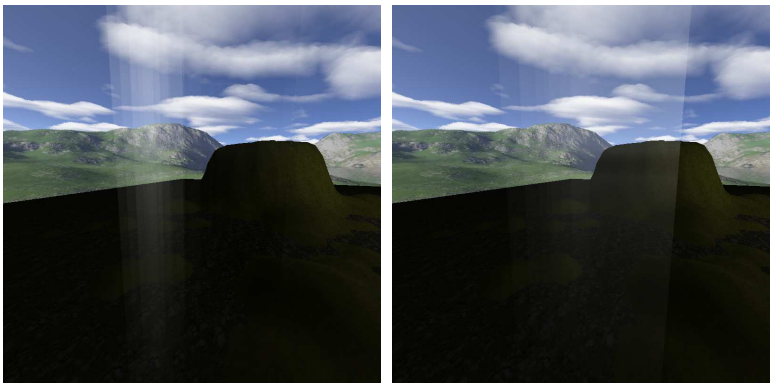

Figure 8.8: A medium built with a single non-null coefficient. Left : all layers of details are summed. Right : details were dynamically removed using our optimization, yielding a coarser result.

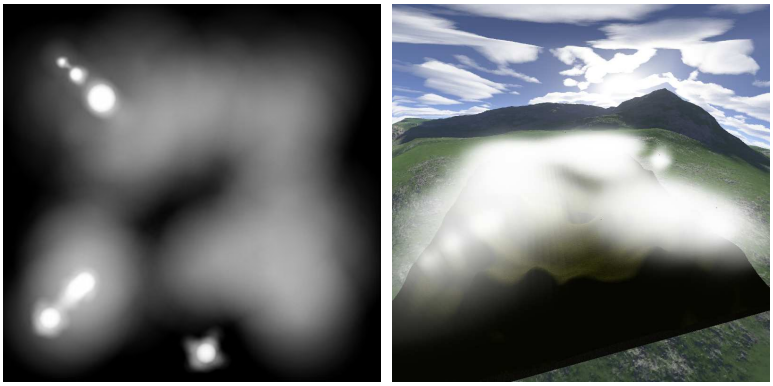

Figure 8.9: Left : a fogmap (64×64) is loaded in our application. Right : the result obtained, modeled using linear B-Splines.

Figure 8.10: More visuals. Left : a 32×32 Haar fog. Right : a 32×32 linear B-Spline fog.

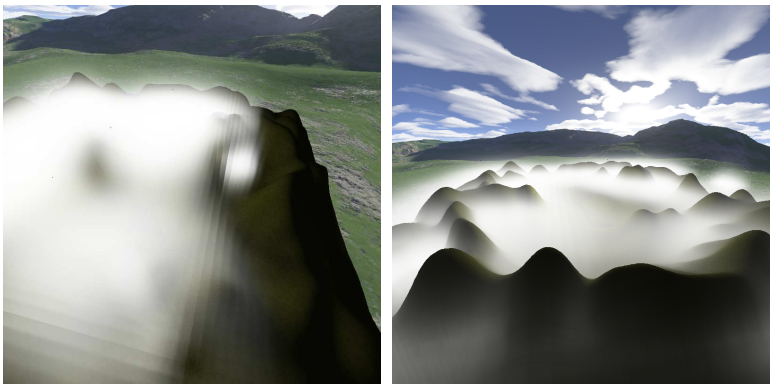

Figure 8.11: More visuals. Left : a 64×64 Haar fog. Right : a 64×64 linear B-Spline fog.

Chapter 9

Optimizations on our fog rendering method

> It may be that mortal free-will can conquer mortal fate; and that going, as we all do, inevitably to death, we go inevitably to nothing that is before death.
>
> Wilkie Collins, *Armadale*

9.1 Modeling the fog's density using a non-regular grid

9.1.1 Overview

In the previous chapter 8, we rendered the medium simply modeled in the structure imposed by the wavelet decomposition :

- 1 basis of 2D scaling-scaling functions $\phi\phi$ (coarse approximation)
- N bases of 2D scaling-wavelet functions $\phi\psi$ (N layers of details)
- N bases of 2D wavelet-scaling functions $\psi\phi$ (N layers of details)
- N bases of 2D wavelet-wavelet functions $\psi\psi$ (N layers of details)

Thanks to the representation based on wavelets, the total number of coefficients contained by all four bases, indeed the number of cells that the ray-marching must process at rendering, remains approximatively the same before and after the wavelet decomposition. All coefficients are only rendered in the worst case, when no part of the medium is far enough from the camera to be discarded.

We could notice several disadvantages of this method :

- Unlike clouds or smoke which feature a denser core, our fog is not opaque enough for our removal of the farthest coefficients to absolutely not be noticeable. In the general case, as the viewer advances into the medium, the frontier between areas with less and more details is slightly visible.

- For a particular frequency (i.e. layer) of details, the coefficients were discarded based solely on their distance (and their frequency) from the camera. Given their size and depth, whether they contribute more or less due to their value was absolutely not considered, which could lead to illogical situations where fully transparent cells are rendered but supposedly more visible non-null cells are discarded.

In this section, we try a new strategy taking advantage of wavelet compression. We first transform the complicated set of summable layers of details output by the decomposition into a simple pyramid of different self-sufficient resolutions. From these multiple resolutions, we generate a non-regular grid of coefficients, where areas within the fog featuring less details are modeled with larger cells. Finally, the medium is rendered based on our ray-marching, where larger cells speed-up the visualization by accelerating the density grid traversal.

9.1.2 Transposing the layers of details into a pyramid of resolutions

9.1.2.1 Expressing the details in $\phi\phi$-bases : overview

What we want to obtain, in the end, is a single function basis modeling the density of the fog in 2D, almost exactly similar to the form under which the fogmap is passed in input to the wavelet decomposition (i.e. a single 2D $\phi\phi$ scaling functions basis), with the difference that the areas in the fogmap where less precision is demanded are modeled with larger cells. At the implementation level, such a data structure will be modeled using a regular grid of the resolution required by the thinnest details at some areas of the irregular grid, and where "bigger cells" are simulated by duplicating a value over several cells of that regular grid that this larger cell covers. However, by also keeping track of the local resolution level at each coefficient, we are able to adapt our ray-marching speed to the virtual size of the cells and only process duplicated cells once.

The first step in generating our non-regular grid consists in reformulating the three pyramids of layers of details generated by the wavelet decomposition so that all this data is expressed under the same form as the approximation. Once the coefficients in all three pyramids of $\phi\psi$, $\psi\phi$ and $\psi\psi$ are expressed as three pyramids using the same $\phi\phi$ basis function, they can all be merged together with the coarse approximation, thus giving one single pyramid of $\phi\phi$ summable layers. Without merging the details and the approximation together, we would have had to render the same number of coefficients, with additional computations required by the manipulation of the four types of data.

In two dimensions, the wavelet framework allows to represent the details of an image using less coefficients than if all those details were modeled in scaling functions bases, with the drawback that four different basis functions are needed to model the whole image ($\phi\phi$, $\phi\psi$, $\psi\phi$, $\psi\psi$). It is obvious that once the different frequencies of details are separated from the approximation, the use of $\phi\psi$, $\psi\phi$ and $\psi\psi$ wavelets to represent these details is not compulsory anymore, everything expressed in terms of $\phi\phi$ functions just results in more coefficients than if using wavelets and also more $\phi\phi$ coefficients than need to model the original unscattered data. However, this problem of a larger number of values to manipulate only exists during the generation of the irregular grid as a precomputation step. At runtime, our grid only features a single level with a resolution changing from one cell to the other, yielding less values than the original fogmap (illustrated in figure 9.2).

9.1.2.2 In 1D : converting ψ_j to ϕ_{j+1} coefficients

We first consider the one-dimensional case for reasons of simplicity. Mathematically, transposing a wavelet basis at level j into the corresponding scaling function basis at level $j+1$ is very straightforward, thanks to the *two-scale relationship for wavelets* (equation 4.7), that we recall here for convenience :

$$\psi(x) = \sum_{k=-\infty}^{\infty} q_k * \phi(2x-k), \tag{9.1}$$

where q_k are the wavelet coefficients of ψ.

The number of coefficients q_k is only infinite in the theorethical definition of ψ, in reality non-zero values are only present in a small delimited interval of k. We remind that k does not start at zero since it does not represent an integer index, but rather an abcissa on the basis function.

This equation shows how to build a wavelet at resolution j from a linear combination of translated thinner scaling functions at level $j+1$. To make this more obvious, this relation can be rewritten directly in terms of ϕ_j, the pattern (untranslated) scaling function at resolution level j, and ψ_j, the pattern wavelet at level j :

$$\psi_j(x) = \sum_{k=-\infty}^{\infty} q_k \cdot \phi_{j+1,k}(x) \tag{9.2}$$

We can also rewrite equation 9.2 in terms of individual translated basis functions $\psi_{j,l}$:

$$\psi_{j,l}(x) = \sum_{k=-\infty}^{\infty} q_k \cdot \phi_{j+1,2l+k}(x) \tag{9.3}$$

Equation 9.3 expresses the direct relationship between the basis functions $\psi_{j,l}$ from the layer of details j and their expression in terms of basis functions $\phi_{j+1,2l}$ used for modeling the approximation at the twice thinner level $j+1$.

On the other hand, the layer of details D_j at level j can be written as :

$$D_j(x) = \sum_{l=-\infty}^{\infty} d_{j,l} \cdot \psi_{j,l}(x), \tag{9.4}$$

where $D_{j,l}$ are the basis coefficients expressing level j's details in terms of wavelets $\psi_{j,l}$.

If we replace $\psi_{j,l}$ in equation 9.4 using 9.3, we have :

$$D_j(x) = \sum_{l=-\infty}^{\infty} d_{j,l} \cdot \sum_{k=-\infty}^{\infty} q_k \cdot \phi_{j+1,2l+k}(x) \tag{9.5}$$

We now have an expression of the whole layer of details j expressed as a function of the wavelet's two-scale coefficients q_k. The two sums are in fact finite : the first sum on l is performed over each coefficient modeling the fog's density details at level j in the wavelet basis, and the nested sum on k is performed over all two-scale coefficients expressing the wavelet in terms of its associated scaling function.

Since our goal is to be able to convert all layers of details D_j and model them in terms of scaling functions ϕ_{j+1}, instead of equation 9.5, what we need to know is, ideally, how to obtain each ϕ_{j+1} basis coefficient $d_{j+1,l}^{\psi\to\phi}$ from the sequence of ψ_j details coefficients $d_{j,l}$. Nevertheless, equation 9.5 generalizes equation 9.1, from the simple definition of a single wavelet in terms of thinner scaling functions to the complete two-scale relationship between the whole signal of a layer of details and those thinner scaling functions ϕ_{j+1}. This already gives us a general overview of the relationship between the representation of a layer of details in the wavelet basis and its modeling in the scaling function basis, that we want to calculate.

Because several scaling functions ϕ_{j+1} participate to modeling a single wavelet ψ_j, by obvious reciprocity, each wavelet gets decomposed over several functions ϕ_{j+1}, whose basis coefficients take the value of the q_k sequence. If we consider that the ratio between the two resolution between levels j and $j+1$ is $1/2$, there is approximatively twice more coefficients in the ψ_j basis than in the ϕ_{j+1} basis, which means that for some types of wavelets where the q_k sequence has more than two values, the definition domains of $\psi_{j,l}$ and $\psi_{j,l+1}$ (spaced by one unit) functions intersect each other. In this situation, each scaling function $\phi_{j+1,m}$ participate, through the two-scale relationship 9.3, to modeling several wavelets $\psi_{j,l}$.

For our problem, by reciprocity, during the decomposition towards level $j+1$ of a layer of details from level j, each ϕ_{j+1} basis coefficient $d_{j+1,m}^{\psi\to\phi}$ that we want to compute will have to accumulate contributions from several ϕ_j basis coefficients $d_{j,l}$. This is exactly the way we implemented this operation.

In one dimension, to transpose a layer of details at level j from a wavelet basis and model it using a scaling function basis at level $j+1$, each coefficient $d_{j+1,m}^{\psi\to\phi}$ is given by :

$$d_{j+1,m}^{\psi\to\phi} = \sum_{k=-\infty}^{\infty} d_{j,\frac{m}{2}-k} \cdot q_k \tag{9.6}$$

9.1.2.3 In 2D : converting $\phi\psi_j$, $\psi\phi_j$ and $\psi\psi_j$ to $\phi\phi_{j+1}$ coefficients

In two dimensions, each level j has not only 1 but 3 layers of details, as function bases using three types of hybrid wavelet functions $\phi\psi$, $\psi\phi$ and $\psi\psi$. Since each hybrid function is a product of two one-dimensional functions, we convert each type of layer of details by applying the two-scale relationship on each axis, either that of the scaling functions 4.3 with the scaling sequence p_k or that of the wavelets 4.7 with the wavelet sequence q_k.

For each type of detail, we have :

$$d_{j+1,m_x,m_y}^{\phi\psi\to\phi\phi} = \sum_{k_x=-\infty}^{\infty}\sum_{k_y=-\infty}^{\infty} d_{j,\frac{m_x}{2}-k_x,\frac{m_y}{2}-k_y}^{\phi\psi} \cdot p_{k_x} \cdot q_{k_y} \tag{9.7}$$

$$d_{j+1,m_x,m_y}^{\psi\phi\to\phi\phi} = \sum_{k_x=-\infty}^{\infty}\sum_{k_y=-\infty}^{\infty} d_{j,\frac{m_x}{2}-k_x,\frac{m_y}{2}-k_y}^{\psi\phi} \cdot q_{k_x} \cdot p_{k_y} \tag{9.8}$$

$$d_{j+1,m_x,m_y}^{\psi\psi\to\phi\phi} = \sum_{k_x=-\infty}^{\infty}\sum_{k_y=-\infty}^{\infty} d_{j,\frac{m_x}{2}-k_x,\frac{m_y}{2}-k_y}^{\psi\psi} \cdot q_{k_x} \cdot q_{k_y} \tag{9.9}$$

Once all three pyramids of layers of details are modeled in terms of 2D scaling functions $\phi\phi$, they can be merged into a single pyramid by simply summing together all corresponding coefficients. Since all layers of details, once expressed in scaling function bases, pass from level j to level $j+1$, there should be no details remaining at level $j=0$, like there were before the conversion. Instead, level $j=0$ only contains the coarse approximation (coefficients a_{0,l_x,l_y}) that we also insert in our final $\phi\phi$-pyramid.

In a nutshell, we have :

$$\begin{cases} d'_{0,l_x,l_y} = a_{0,l_x,l_y} & \text{for the approximation at } j=0 \\ d'_{j,l_x,l_y} = d^{\phi\psi\to\phi\phi}_{j,l_x,l_y} + d^{\psi\phi\to\phi\phi}_{j,l_x,l_y} + d^{\psi\psi\to\phi\phi}_{j,l_x,l_y} & \text{for the details from } j \geq 1 \end{cases} \tag{9.10}$$

9.1.2.4 From layers of details to multiple resolutions

At this step, we have transposed all three types of details generated by the 2D wavelet decomposition from being represented in four different pyramids of function bases, to an unified representation in a single pyramid of $\phi\phi$ scaling function bases. The coarse approximation, already represented in terms of $\phi\phi$ functions, was also imported in this pyramid, and inserted at level 0. We can name this pyramid of details P_{det}.

Parallely, as part of the algorithm for the generation of our non-regular grid, we need to prepare another instance P_{app} of our pyramid P_{det} where we "flatten" all data. This consists in replacing each layer of details by an approximation with the same resolution.

To achieve this, we start from the coarsest approximation at level 0 and simply advance level by level where we alternatively accumulate each layer of details and replace it with the current result. To add two consecutive levels j and $j+1$, we use the two-scale relationship for scaling functions (equation 4.3) to express level j in terms of functions $\phi\phi_{j+1}$ and sum up each two corresponding coefficients.

9.1.3 Generating the non-regular grid

9.1.3.1 Determining where to place larger cells

We now consider again our pyramid of summable details P_{det}. To parameterize the propensity of the algorithm to eliminate the least significant details and create larger coefficient cells, we define a threshold d_ϵ, as the value below which a basis function coefficient can be considered null and therefore be discarded. When $d_\epsilon = 0$ no details are discarded, and more and more details are discarded as d_ϵ increases.

If we consider two consecutive levels of our multiresolution pyramids, there is a ratio of 2 between the resolution at level $j-1$ and that of level j. In P_{app}, this means that one cell, indeed one coefficient of the approximation at level $j-1$ overlays, in the pyramid, four twice thinner coefficients from level j, sixteen cells from level $j+1$, etc. Therefore, in order to be able to replace a group of small cells from the maximum resolution level N by a larger cell imported from level $N-1$, all four cells that this larger cell overlays have to be discardable, otherwise simplifying the grid by downgrading the resolution in this area is not possible.

For each of the thinnest cells of the maximum resolution level N, we begin by determining how many l levels of details could be discarded so that, ideally, the resolution could be locally reduced

from N to $N-l$. At this step, we only consider each cell from level N separately, without checking, for the moment, that other 4^l cells of the same group also fulfill the neglectability condition.

In a second pass, we visit all levels from N to 0 and sequentially repeat the same process to account for the group of cells. At level j, we iterate on each group of four cells on which a coarser cell from level $N-1$ superimposes and, when the maximum significant resolution level of all four cells is lower than j, we know that the resolution can be downgraded to level $j-1$, and these four cells symbolically replaced by the corresponding twice larger cell from that level. When all levels are processed, we possess the crucial information of which resolution level is associated to each position in the medium, we are therefore ready to build the final data structure.

9.1.3.2 Creating the graph

For obvious memory layout reasons, the non-regular grid is actually implemented as a regular grid with the same dimensions as the thinnest resolution level N. If we consider the small number of coefficients modeling the fog's density (i.e. less than 100×100), concerns about memory savings appear far from relevant. However, our goal is to speed-up the rendering in comparison with a direct visualization of the uncompressed fogmap on the basis that larger cells will be traversed faster by the ray-marching.

To optimize the grid traversal, our data structure resembles more a graph, with each coefficient cell being provided a list of links to its direct neighbours. These links are transmitted to the GPU as a texture, and contain some precomputations in the perspective of the grid traversal, including some elements which in our previous method were computed at runtime such as the indexes of the next cell traversed, the texture coordinates in the coefficients texture, etc.

Figure 9.3 shows some results obtained with our non-regular grid, for different values of d_ϵ. With Haar wavelets, the result, seen from above, shows hard transitions, which are not really wanted in terms of quality, but are interesting to check the differences in the size of the cells.

Figure 9.1: Example of very coarse irregular grid. Using Haar wavelets triggers sharp visual transitions, making it possible to distinguish the different cells.

9.2 Rendering the medium's coefficients separately

9.2.1 Overview

In this second attempt to optimize the visualization of our medium, we try to discard all coefficients which have a neglectable value by proceeding in a completely different manner than above. With the non-regular grid, our major constraint that limited the number of coefficients which could be dismissed was essentially the grid-like nature of the data structure, and the rendering algorithm articulated around the ray-marching through that grid. Indeed, the iterative nature of the ray-marching imposes to visit all cells along the way, and there is no way, even if we know which cells can be dismissed, to directly jump from one cell to a cell which would not be one of its direct neighbours. For this reason, there remained a noticeable number of poorly contributing cells in the non-regular grid which could not be discarded, either because such cells were of the lowest resolution level $j = 0$ so that it was not possible to create larger cells, either because they were part of a group where some other cells where not discardable so that the resolution could not be coarsened.

To solve this problem, instead of passing the whole density distribution to the GPU and visualizing it based on a ray-marching in a single pass, we consider rendering each significant coefficient separately and accumulating the result on the framebuffer, without the need of traversing the grid from the camera to the nearest object.

On the CPU, as a preprocessing step after the decomposition, we analyse the coefficients from the approximation and the three pyramids of details, and simply build a list of all significant coefficients. In comparison with the previous strategy, since each grid cell is processed completely separately from its neighbours, they can all be discarded even on the root level.

9.2.2 Rendering

To visualize the medium, we activate additive blending and loop over the list of coefficients. For each value, we draw the boundaries of its cell with simple quads. We also activate the culling on front faces, so that the shader is executed on the back of the polygon, the fragment's position directly indicates the farthest intersection between the view ray and the cell. If we instead render the front face, the problem arises that when the camera stands inside the cell, all its boundary quads are seen from the back and therefore not visible, which is necessary for the shader to be executed.

In the shader, from the fragment's position taken as the cell's exit intersection, we deduce the entry position where the ray intersects the culled front face of the cell's polygon. The cell's density is not read from a texture anymore, but updated as an uniform value. Based on both the entry and the exit positions within the cell, we extract the corresponding integral from our precomputations and multiply it with the coefficient's value.

To account for the occlusions by the geometry, before rendering the medium, we first render the geometry and get the depth buffer, which is transmitted to the GPU. In the shader, we fetch the pixel's depth value, compare it with the cell exit distance, and if needed we rectify the exit position by replacing it with the position of the surface. If the deduced entry position lies behind the corrected exit position, we know that the whole cell is hidden by the geometry, and we discard it immediately.

Non-zero threshold d_ϵ	0.0	0.2	0.4	0.6	0.8	1.0	2.0	3.0	4.0
Non-regular grid (fps)	24	24	25	26	29	30	33	35	36
Separate coefs (fps)	19	21	22	22	23	24	26	29	32

Table 9.1: FPS comparison between our two optimizations. GPU : NVidia GeForce 310M, fog is Haar 32×32, screen resolution is 800×600. The same fog is rendered by our standard method at 24 fps for $\mu = 1.0$ (full details), and 33 fps for $\mu = 0.0$ (approximation only), if we take $d_{\text{ref}} = 7.0$.

Based on the inclination of the view ray, we compute a vertical attenuation coefficient to rectify the horizontal integral and account for the exponential fading of the medium's density vertically. As the fragment's color, we output the three components of the medium's color as defined by the user, and weight it using the alpha channel by the integral of the density as traversed by the view ray between the entry and the exit intersections with the cell. When all coefficients are processed, the framebuffer contains the integral of the density traversed by the view ray between the camera and the nearest geometry.

9.2.3 Results

Figure 9.2 summarizes our two optimization attempts. In (A), without going further, we have absolutely no qualitative information about the content of the grid, therefore all grid cells must be traversed and rendered. In (B), a wavelet decomposition is performed, which gives us more information about the content of the density map, i.e. which areas require a higher resolution than others. In (C1), describing what we propose in section 9.1.3, use the frequency information brought by (B) to locally decrease the resolution in areas of the grid featuring less details. The major drawback is, since the rendering is still based on a grid traversal, all cells must be rendered, even if their coefficient is zero. In (C2), we modify our rendering algorithm to avoid having to perform a grid traversal. We build a list a all non-zero coefficients, and render them separately, which is described in section 9.2.

We are still currently working on these optimizations. Table 9.1 provides some results for different values of the non-zero coefficient threshold, and compares the performances of the two methods. We tested our implementation of these optimizations on a NVidia GeForce 310M GPU.

First of all, the non-regular grid provides good results. When $d_\epsilon = 0$ and the grid is at its maximum resolution everywhere, the non-regular grid rendering algorithm is performed at the same speed as a simple ray-marching on all levels. As the threshold increases, "larger" cells are generated in the less detailed areas, which speeds-up the rendering. With this fog, for instance, we noticed that we had to increase d_ϵ up to 4.0 for all details to be discarded, for which we measured 36 fps against 33 fps for our original method with $\mu = 0$.

We can therefore conclude that our solution using a non-regular grid provides a small gain in performance.

However, we were a bit disappointed by the results obtained by our second optimization attempt, by rendering all non-zero coefficient separately. We think that what makes it slower than previous approaches is the way we draw the quads, one by one. We still believe that this strategy is promising, and we are working at improving and optimizing our implementation, using, for example, vertex buffer objects.

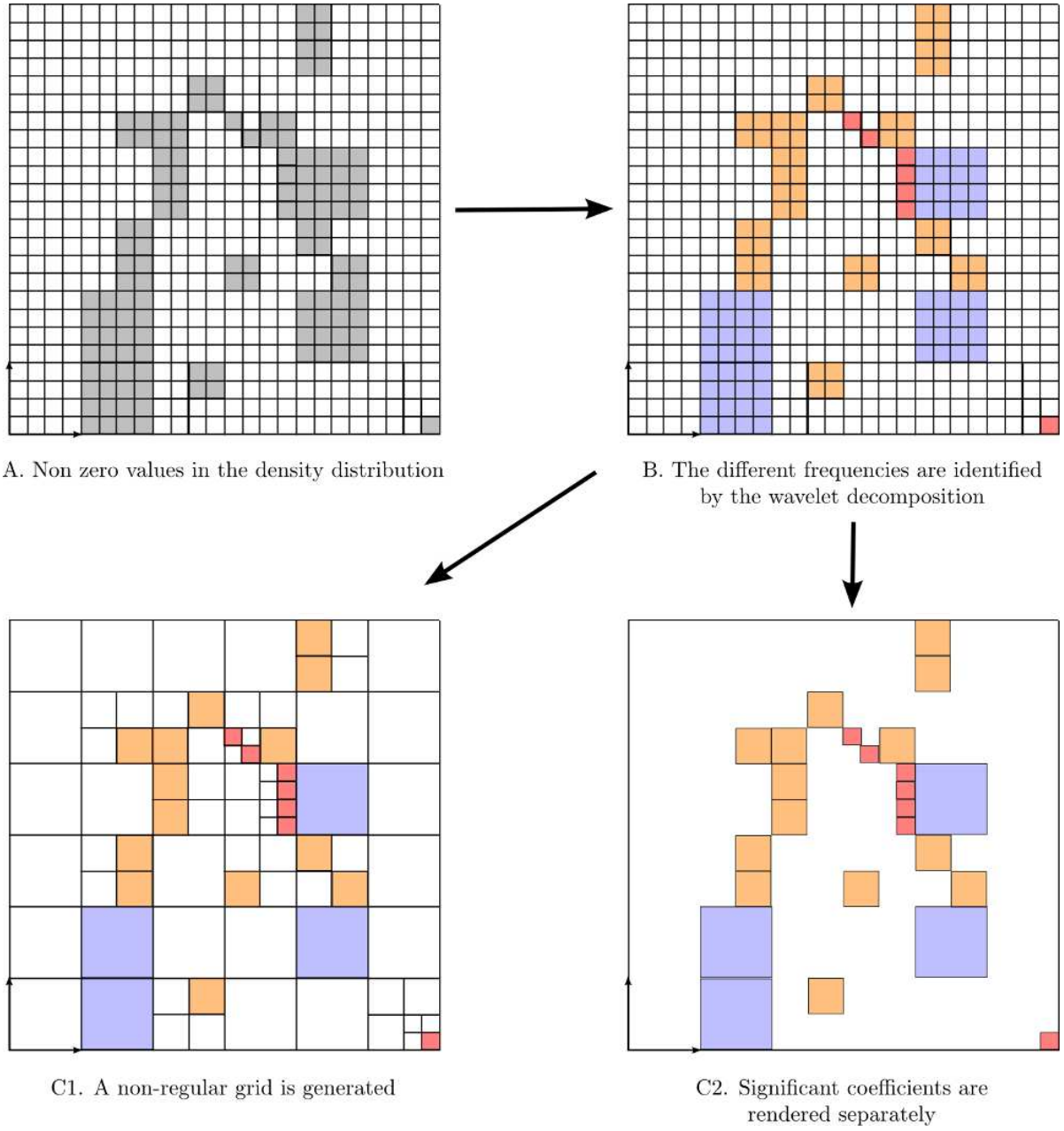

Figure 9.2: Overview of our two attemps to optimize our fog rendering method. In (A), the fogmap is loaded, some coefficients are null while others (in gray) bring a significant contribution and cannot be discarded. In (B), the different frequencies of details are identified by the wavelet decomposition. In (C1), our first idea was to generate a non-regular grid, featuring larger cells in areas featuring lower frequencies. In (C2), we chose to completely modify our visualization process, and render each coefficient separately.

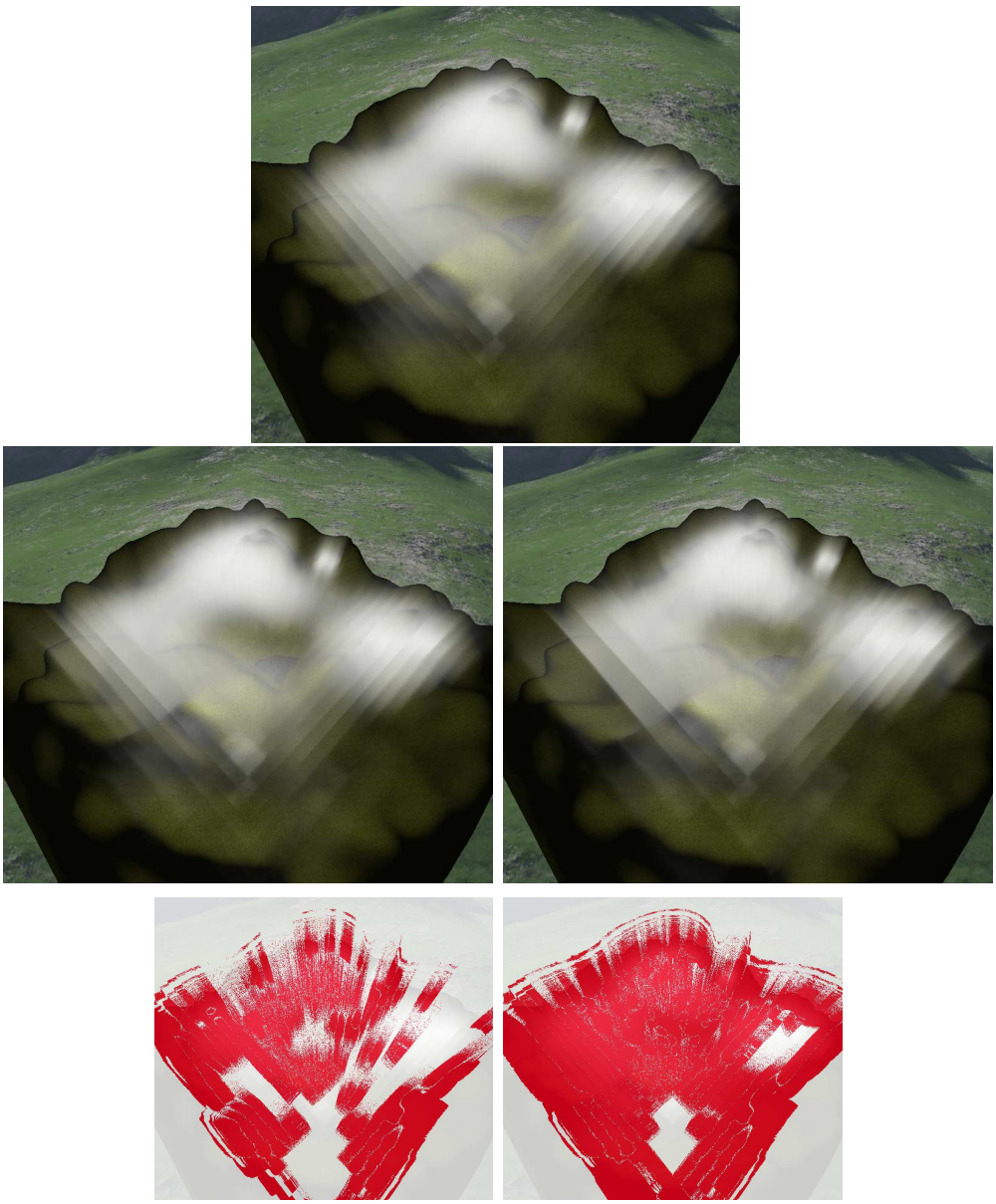

Figure 9.3: Non-regular grid : quality comparison with a 32×32 Haar fog, with a screen resolution of 800×600, on a NVidia GeForce 310M. top : $d_\epsilon = 0.0$ / 17 fps, middle left : $d_\epsilon = 0.6$ / 19 fps (RMS error : 0.85%), middle right : $d_\epsilon = 1.0$ / 22 fps (RMS error : 2.15%). The bottom images represent their respective difference with the complete untouched density at the top.

Chapter 10

Modeling and animating heterogeneous fog using chaotic maps

> "Men's courses will foreshadow certain ends, to which, if persevered in, they must lead," said Scrooge. "But if the courses be departed from, the ends will change. Say it is thus with what you show me!"
>
> Charles Dickens, *A Christmas Carol*

10.1 Introduction

The goal of Computer Graphics is to represent realistic virtual universes. When we look at the real world, we can notice that chaotic and fractal behaviours are very common in nature, that is why researches in Computer Graphics and dealing with the representation of natural phenomena frequently chaos theory to model natural processes. Chaotic processes can be compared to Darwin's theory of Evolution : generating complexity through a large number of iterations of very simple random processes. Indeed, the main advantage of chaotic equations lies in their simplicity. When repeating chaotic processes many times, we can observe a continuous evolution on the result, potentially leading to very sophisticated random features making objects look more natural.

We follow this idea in an exploratory research where we apply chaotic processes to obtain outdoor fogs looking more natural. Using Julia fractals, our method first generates a two-dimensional chaotic *fogmap*, which represents the distribution of the medium's density in the scene. We then pass this chaotic fogmap in input to our real-time rendering pipeling to visualize the result.

Thanks to the chaotic map, we can animate continuously the fog by changing the initial parameters, such as the c constant in the complex quadratic maps (Julia set). We also consider iterated functions related to fluids dynamic to represent the fog phenomenon more closely.

10.2 Mandelbrot and Julia sets

10.2.1 Foreword

Julia sets are named after Gaston Julia (1893 - 1978), who conducted researches on the theory of iterations over rational functions on the complex plane, mostly during the 1917-1918 period, and published in his *Journal of Pure and Applied Mathematics*. His works only became known to the general scientific community after being quoted by another mathematician, Benoît Mendelbrot (1924 - 2010), in his famous work *Fractals : Forms, Chance and Dimension*, published in 1977 and in which he first coined the word *fractal*. Another researcher who, parallely, contributed much to non linear iterations in the complex plane is the French mathematician Pierre Fatou (1878 - 1929).

10.2.2 Interating on complex polynomials

10.2.2.1 Iteration on Mandelbrot and Julia sets

Both Mandelbrot and Julia sets of fractals are based on the same equation :

$$z_{k+1} = z_k^2 + c, \text{ with } k \in \mathbb{N}, \text{ and } z_k, c \in \mathbb{C} \tag{10.1}$$

This is the equation of an iteratively evolving 2D shape, also parameterized in 2D as a function of all points of the complex plane. Actually, Mandelbrot and Julia curves are parameterized differently, and this is the reason why their shape and evolution are very distinct. This equation defines the value z_{k+1} at the next iteration $k+1$ of the evaluated point as a recursive function of its value at the current iteration k and another complex parameter c.

To generate the popular very coloured Mandelbrot and Julia fractal images, the value z_k associated to each pixel is computed at several consecutive iterations $\{z_0, z_1, ..., z_n\}$. The speed with which the values run away from the origin decides the color of the pixel, the black central shape corresponding to points whose values do not run away but become stable after a few iterations, or get caught within a stable cycle of two or three iterations.

10.2.2.2 The Mandelbrot set

The Mandelbrot set is defined in the parameter plane, i.e. the different positions of z_k are obtained by replacing the parameter c by each point on the plane, for which we want to compute a pixel value. The overall aspect of the set obtained depends in the initial value z_0.

10.2.2.3 The Julia sets

Julia fractals are obtained in a similar way, but the role of z_0 and c is swapped. Indeed, z_0 is replaced with the different points on the complex plane, and the overall appearance of the sets depends on the parameter c. We say that Julia sets are defined on the *dynamical plane*.

10.3 Obtaining a chaotic *fogmap*

In our implementation, we use Julia sets, but the Mandelbrot set can be used as well. To model our fog using a chaotic density distribution, we apply the same process as when generating a coloured

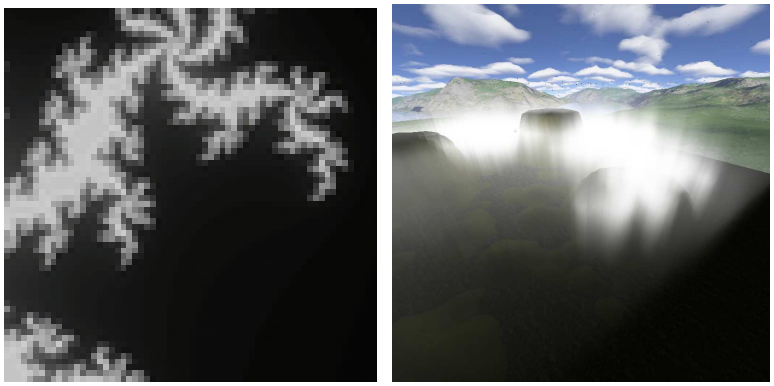

Figure 10.1: Left : a chaotic fogmap (32×32) generated from julia sets. Right : the fog obtained, using Haar wavelets.

fractal image. When looking at fractal images generated from Mandelbrot or Julia sets, we can notice, in the case that the set is connected, the presence of a stable core, surrounded by belts of different colors representing unstable points running away towards infinity at different speeds. Instead of a color, the diverging speed of each point decides its density in the fogmap : the highest density is affected to stable points, and as the z_k tend to faster and faster to infinity, their density tends faster and faster towards zero.

Modeling the density distribution of fog never requires a high resolution, therefore there is no need to compute fractal images larger than for example, at the maximum, 100×100. Moreover, we do not generate the fogmap from an overall image of the whole set, otherwise the result look too binary, passing from fully-opaque coefficients at the center to completely invisible coefficients outside the main shape. Instead, the fog's chaotic density is generated from a zoomed portion on the border of the stable area, where we can maximize the chances of rendering points which are unstable, therefore not completely opaque, and whose divergence speed smoothly varies over the whole rendered area (see figure 10.1). In a nutshell, interesting chaotic density distributions are mostly obtained by rendering portions of the sets where both quantities of stable and too unstable points are minimized.

The medium can be animated by changing the value of the set's parameter c, as illustrated in figure 10.2. Although small chaotic maps, e.g. 64×64 can be generated in real-time at each frame, when using our optimization based on the extraction of different resolutions of details, depending on the number of decomposition steps applied on the fogmap, the precomputations can significatively influence the performances. In our implementation, we choose to prepare a series of precomputed and pre-decomposed density distributions, through which we cycle at rendering.

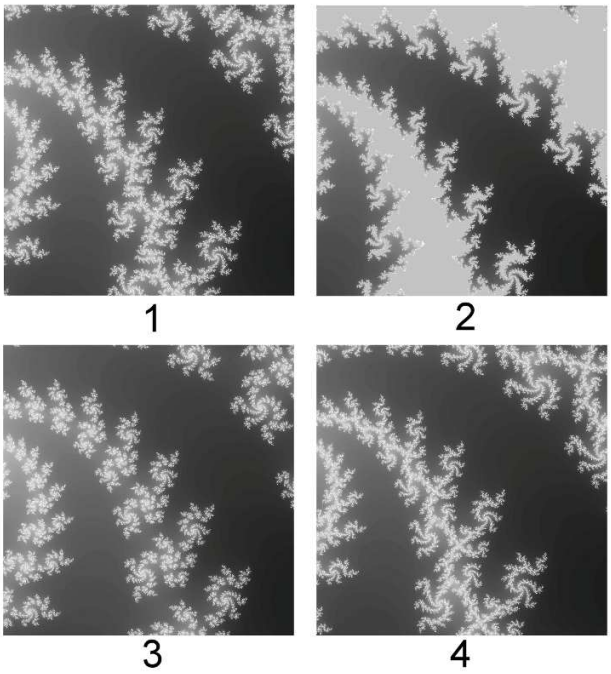

Figure 10.2: Four states of an animated *fogmap* generated using Julia sets.

Chapter 11

Beginning the illumination of our fog

> Man is not the creature of circumstances.
> Circumstances are the creatures of men.
>
> Benjamin Disraeli, *Vivian Grey*

Note : Because of a lack of time, we unfortunately could not finish the development of this part before submitting this manuscript for review. However, we are confident to be able to present more results at the date of the defense.

11.1 Overview

In this chapter, we begin to illuminate our fog by considering single-scattering and a single light source, the sun. We handle occlusions by the geometry based on a shadow mapping, and also handle shadows by the medium on the geometry. We also implemented an attenuation of direct lighting due to optical depth as sun rays pass through the fog, but we only consider, for the moment, an analytical attenuation due to the cell itself. Indeed, rays emitted by the source in the direction of a cell in the horizontal medium, where they will be scattered towards the camera, are not yet attenuated by cells other than the one where the scattering event occurs.

11.2 Illumination model

Similarly to our first method presented in section 8.2, we start with equation 8.1, the complete model that we recall here for convenience :

$$L(O) = \tau(O,P)\ L(P) + \int_{u=0}^{|OP|} \tau(O,x(u))\ \rho(x(u))\ \beta(x(u))\ F(\alpha)\ J(x(u))\ du \tag{11.1}$$

At this step in the development, we assume an isotropic scattering, where $F(\alpha) = 1$.
We finally use the following model :

$$L(O) = \tau(O,P)\ L(P) + \int_{u=0}^{|OP|} \tau(O,x(u))\ \rho(x(u))\ \beta(x(u))\ J(x(u))\ du \tag{11.2}$$

11.3 Direct lighting

We assume a single light source, the sun, that we suppose always situated outside the medium. In the litterature [Max86] [NMN87] [KONN91] [LMAK00], because of its huge distance from the Earth, the sun is usually represented as a directional source. In our method, we use the shadow mapping technique to compute occlusions by the geometry, which involves rendering the scene from the sun's point of view. For this reason, the sun must be affected with a true finite position within the scene, even if we use an orthographic projection to simulate parallel sun rays.

To approximate the incident light $J(x(u))$ arriving at position $x(u)$ on the view ray, a more accurate solution would consist in a numerical integration based on a ray-marching between $x(u)$ and the source's position S. Instead, for the moment, we only implemented a quick approximation that we hope to be able to correct before the date of the defense.

Vertically, the analytical definition of $\rho^Y(x(t))$ enables an exact analytical integration of the vertical attenuation (8.8) :

$$\begin{aligned}\int_{u=0}^{|x(u)S|} \rho^Y(x(u))\ \mathrm{d}u &= \int_{u=0}^{|x(u)S|} \gamma^{|\mathrm{fog}_y - x(u)_y|}\ \mathrm{d}u \\ &= \frac{\gamma^{|\mathrm{fog}_y - x(u)_y|} - \gamma^{|\mathrm{fog}_y - S_y|}}{\ln(\gamma)}\end{aligned} \tag{11.3}$$

Horizontally, we assume that, on the one hand, the fog is relatively thin vertically in comparison to its large scale and, on the other hand, the sun is supposed to be situated above the fog. This means that the incident sunlight will always have a strong vertical direction when arriving on the fog, and will not be attenuated by much more than one cell, i.e. the cell where the scattering event occurs.

In these conditions, $\tau(S, x(u))$ can be approximated by taking the mean value of the density on the current cell V traversed in the grid, written κ, and multiplying it with equation (11.3), the integral of the vertical component of the density :

$$\tau(S, x(u)) \simeq \exp\left[-\kappa\ \eta\ \int_{t=0}^{|x(u)S|} \rho^Y(x(u))\ \mathrm{d}u\right], \tag{11.4}$$

with κ the mean value of the integral of the density along cell V :

$$\kappa = \frac{\displaystyle\int_{x(u)\in V\cap OP} \rho^{XZ}(x(u))\ \mathrm{d}u}{|V \cap OP|}, \tag{11.5}$$

Since ρ^Y is only an one-dimensional function, we apply a factor η, necessary to account for the possible inclination of light rays :

$$\eta = \frac{|S\ x(u)|}{\sqrt{(x(u)_x - S_x)^2 + (x(u)_z - S_z)^2}} \tag{11.6}$$

11.4 Rendering shadows

To take into account occlusions by the geometry, we chose to use the shadow mapping algorithm, for two main reasons :

- It is well-adaptated to scenes featuring a single light source, situated far and above all objects.
- It is natively adaptated to the shadowing of solid surfaces as well as participating media. Since we know, from the shadow map, from which distance to the light source objects are shadowed, we can indifferently compare the depth of a true fragment (actually generated by the pipeline from a surface) and that of any theorethical position (not linked to OpenGL geometry) within a volume.

However, shadow mapping also presents notable disadvantages, such as the need for a high-resolution shadow map in order to limit aliasing effects. Even if several solutions have been proposed to reduce this kind of visual artefacts, such as *Cascaded Shadow Maps*, when we have more time, as a perspective, we directly plan replacing the use of shadow mapping with *shadow volumes* [Cro77], used in the early volume rendering methods, which do not involve rendering the scene from the source and is therefore not subject to aliasing.

Shadows by the medium are simulated by computing the projection of the fogmap on surface fragments. This simple technique seems to already provide rather acceptable results, the only problem, in comparison with a true shadowing method computing the optical depth traversed by direct lighting rays, is that, for the moment, a surface situated within the fog would be either completely lit, or completely shadowed.

11.5 Preview

We finish with some early in-development visuals.

Figure 11.1 shows the realism added to the scene when shadows by the geometry are computed.

Figure 11.2 shows the scattering of yellow light by the medium. The lower part of the layer of fog is very dark, which corresponds to the true illumination of the scene. The walls of the skybox are simply textured, and their brightness is not related to the illumination of the scene.

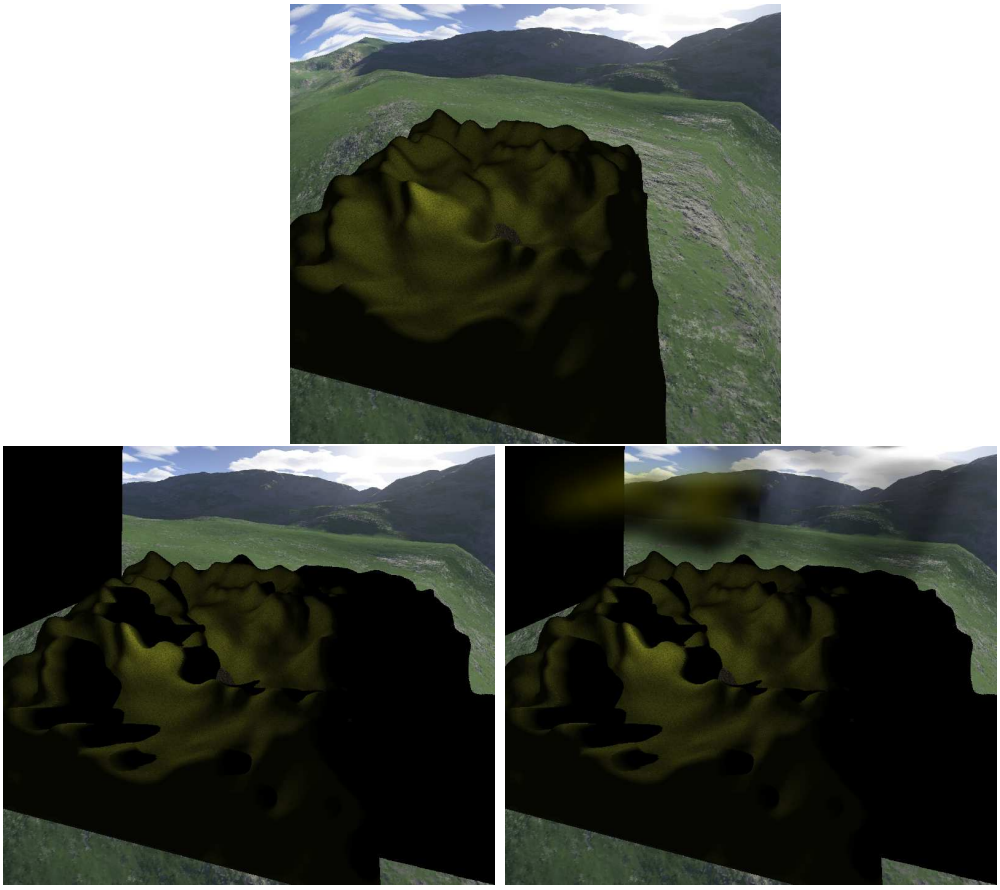

Figure 11.1: Some visuals at the current state in the development : scene with no medium or shadows (top), shadow mapping is activated (left), the medium is present, illuminated by a yellow light source (right).

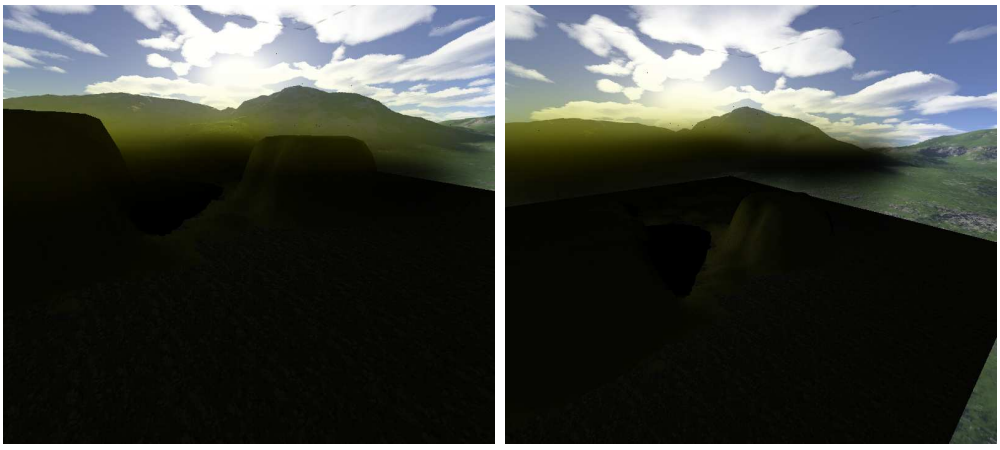

Figure 11.2: Closeups on the scattering medium.

Chapter 12

Illuminating and rendering single-scattering media in real-time using occlusion propagation

"What do we live for, if it is not to make life less difficult to each other ?"

George Eliot, *Middlemarch*

12.1 Introduction

12.1.1 Foreword

Participating media are massively used nowadays, both in applications where real-time is required, such the video game industry and in interactive simulations, as well as in domains where visual quality is much more important than user interactions, like cinema and animation.

Although displaying surfaces is now well-mastered, rendering phenomena such as clouds or fog is incomparably less trivial. However, handling natural phenomena is now absolutely compulsory for any rendering engine, since these are the elements in a scene which contribute most to photorealism.

12.1.2 Motivation

This work was motivated when noticing that, in 2010, we could not find any simple real-time single-scattering illumination method based on point light sources and able to render occlusions due to both solid geometry and participating media in an unified fashion, just like [EP90] did in their time, using the A-buffer.

In [MHLH05], Magnor et al. propose a method to visualize reflection nebulae in interactive time. Similarly as our method, the medium is first discretized into a set of voxels, then illuminated in a precomputation step, before being finally rendered at runtime. The method handles single scattering as well as local per-voxel multiple scattering, but the density within the nebula is kept fixed, due to the precomputation step.

In [ZRL+08], Zhou et al. manage to simulate occlusions between objects and the medium, but illumination is only based on low-frequency environment lighting, which removes substantial constraints, since they can assume that the source will never lie within the medium. They approximate light blocking geometry by RBF particles, used to model the medium. This is simply equivalent to transforming all solid geometry into smoke, which can nevertheless provide acceptable smooth shadows looking plausible due to the use of low-frequency lighting.

In the famous Fogshop method [ZHG+07b], Zhou et al. illuminate the same type of medium using point light, but do not handle occlusions by the geometry. They propose an approximation to the computation of optical depth between the source and each medium particle which, in order to limit visual shadowing errors, imposes to only use small particles, even when modeling large media.

In [RZLG08], Ren et al. illuminate smoke and are able natively to compute shadows cast by the medium on the surfaces, but only suggest to use shadow mapping to also account for occlusions by the geometry, thereby prohibiting the insertion of light sources within the medium.

In 2009, Kaplanyan [Kap09] introduces the concept of Light Propagation Volumes, to scatter indirect lighting. The technique has been implemented in the CryEngine3 video game engine. After generating reflective shadow maps and obtaining a set of virtual point lights on reflective surfaces, direct lighting is injected in a radiance volume, which is a simple volumetric grid. In a third step, using graphics hardware, indirect lighting is propagated from cell-to-cell by iteratively solving differential schemes inside the volumetric grid.

Although focusing only on indirect lighting on surfaces, this approach by light propagation within a volumic grid is fast, allows more flexibility and could be used to scatter direct incoming radiance within a scattering media as well, as a more flexible and GPU-friendly way of performing the first step of [KVH84].

12.1.3 Overview

We are able to simulate shadows by the medium and the geometry, onto both types of elements. The illumination is parameterized using point light sources, which can illuminated the single-scattering medium from within, and generate such effects as glows. Our implementation handles up to eight point light sources, but the method in itself is not limited.

The contribution of this work is :

- Establishing a modular framework to illuminate and render inhomogenous scattering media.
- Introducing a new approach for the precomputation of the optical depth between light sources and each point in the scene.
- Presenting an efficient implementation of this framework, fulfilling real time expectations.

12.2 Theoretical background

12.2.1 Modeling the participating media using a radial function basis

Because our participating media is not static and can evolve over time, the modeling step must be as simple as possible for the user.

Like [ZHG+07a], we choose to model our heterogeneous participating media as a sum of radial basis functions (RBF). RBFs, which can be considered as a set of particles, appear very convenient when modeling media such as smoke or fog.

To define the medium's appearance, the user just provide an unsorted list of radial particles, which can differ in both amplitude and scale. The particle's density will then be evaluated and injected into a volumic grid. Even if our method is very modular, other models can be used as well, provided that the density injection step is properly adapted.

As the radial function itself, we simply chose the gaussian function, defined on $\mathbb{R}^d$:

$$\beta(x) = ce^{-a^2\|x-b\|^2} \tag{12.1}$$

where $a \in \mathbb{R}$ is its amplitude, $b \in \mathbb{R}^d$ its center and $c \in \mathbb{R}$ its scale.

Given equation 12.1, evaluating a Gaussian in $\mathbb{R}^d$ where $d \in \mathbb{N}$ is as straightforward, since β remains an one-dimensional function of the distance between position x and center b, whatever the dimension of the space.

To evaluate a function defined in a radial function basis, we must sum all basis functions that overlay at the given coordinates.

$$f(x) = \sum_{i=0}^{N} \beta_i(x) \Longleftrightarrow f(x) = \sum_{i=0}^{N} c_i e^{-a_i^2\|x-b_i\|^2}, \tag{12.2}$$

where i is the index of the RBF, and N is the total number of RBFs in the basis.

12.2.2 Our illumination model

The appearance of a participating medium is linked to airlight (see [Arv93] for further details). When light is emitted from a point light source S, then goes through a participating medium with an extinction function f (see figure 12.1), the light $L_S(O)$ received by the observer at position O, and who looks in the direction of point P is given by :

$$L_S(O) = \int_0^{|OP|} \rho(x(u))F(\alpha)\beta(x(u))\frac{J(S)}{\|x(u)S\|^2}e^{-\tau(O,x(u))-\tau(x(u),S)}du, \tag{12.3}$$

where $F(\alpha)$ is the scattering phase function, $\rho(x(u))$ is the medium's density, α the scattering angle, $\beta(x(u))$ is the absorption coefficient, $J(S)$ the intensity of light S, and $\tau(A, B)$ is the optical depth of the medium between points A and B :

$$\tau(A, B) = \int_0^{|AB|} \rho(x(t))dt \tag{12.4}$$

In our method, we will assume isotropic scattering and no absorption, yielding :

$$F(\alpha)\beta(x(u)) = \frac{1}{4\pi} \tag{12.5}$$

Therefore, our final model is :

$$L_S(O) = \int_0^{|OP|} \rho(x(u))\frac{1}{4\pi}\frac{J(S)}{\|x(u)S\|^2}e^{-\tau(O,x(u))-\tau(x(u),S)}du \tag{12.6}$$

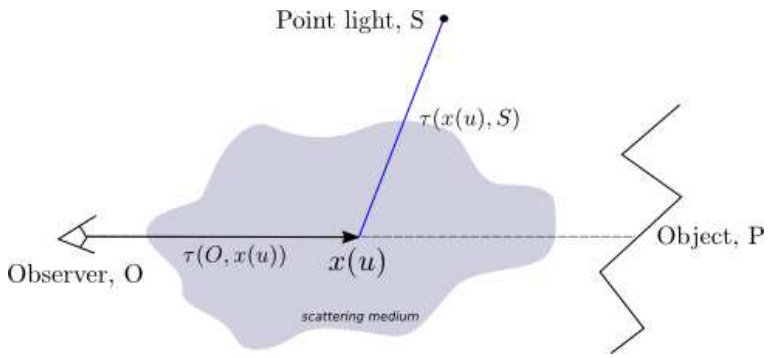

Figure 12.1: The integral of $\tau(x(u), S)$ is computed during the occlusion propagation stage (blue path), between each light S and each position $x(u)$ along the view ray. The integral of $\tau(O, x(u))$ is accumulated at the rendering step.

12.3 Our method

12.3.1 Overview

Our pipeline is mainly composed of four steps :

1. Density injection : The medium's extinction function ρ is injected into a 3D grid, called the *extinction volume* (EV).

2. Geometry voxelization : All solid surfaces in the scene are transformed into voxels, which are then injected into the extinction volume, with their normal vectors.

3. Occlusion propagation : The optical depth $\tau(S, x(u))$ is integrated from each light source S to each cell in a 3D grid called the *occlusion propagation volume* (OPV).

4. Volume rendering : The participating medium is rendered, using a simple ray-marching technique, and based on data obtained from the two previous steps.

The following sections describe all these steps, explain the algorithm and give details about the implementation. Our application is programmed using C++ and OpenGL, and GLSL for the fragment shaders.

12.3.2 Density injection

12.3.2.1 The extinction volume

Due to the low-frequency nature of the scattering medium, a simple technique based on ray-tracing would need to evaluate the extinction function several times at the same position. For each voxel in the light propagation volume, we can precompute an associated extinction coefficient and store it in a separate grid, the extinction volume. Each coefficient can then be accessed as much as needed by simply fetching the value from a 3D texture. All further radiance computation will be based on the resolution of the occlusion propagation volume, therefore we will never need a higher resolution for the density.

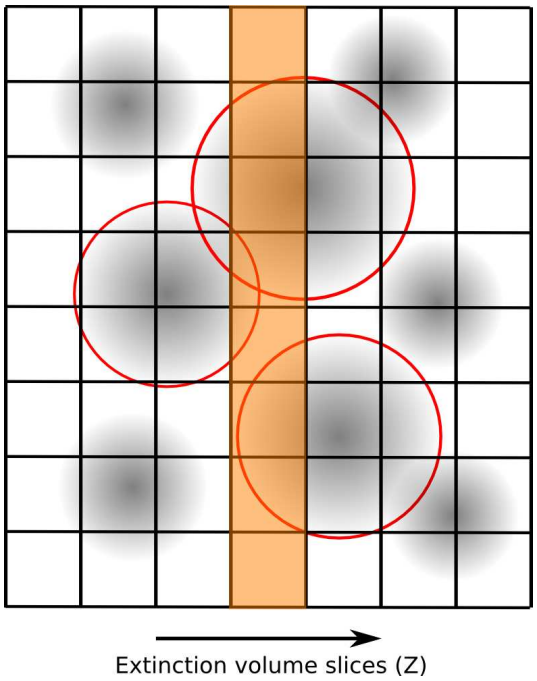

Figure 12.2: Planesweep along texture slices. Gaussians shown in red contribute to the slice shown in orange.

12.3.2.2 Density injection algorithm

The extinction volume is then stored in a 3D texture, which has the same dimensions than the occlusion propagation volume, and the same position in the scene. For each texel, we only need to store one decimal value, therefore a 16-bit floating point encoding is sufficient.

To fill the texture, we perform a plane sweep along the Z-axis (depth) (see figure 12.2). We setup a simple parallel projection, and modify the viewport to fit the extinction volume's width and height. Each plane sweep step fills one texture slice, which is bound to a framebuffer object.

First, the unsorted RBF list, initially provided to shape the medium, must be ordered according the particles' Z-coordinate. To keep track of the bounds of the RBFs list section delimiting the particles which contribute to the current slice, we simply use two iterators which are advanced properly at each new step. For each RBF contributing to the current slice, we draw its bounding quad facing the camera. The RBF is computed at each point on the quad using a pixel shader, and since several gaussians may be overlayed on the same slice, we need to activate additive blending.

Based on equation 12.1, in three dimensions, if $X(x, y, z)$ is a point in the space, we have :

$$\rho(x(u)) = \sum_{i=0}^{N} c_i e^{-a_i^2[(x-b_{i_x})^2+(y-b_{i_y})^2+(z-b_{i_z})^2]} \tag{12.7}$$

where K_t is the extinction function, and N is the total number of RBFs composing the medium.

Algorithm 10 shows the pseudo-code for the main planesweep performed on the CPU. The pixel shader computes only the evaluation of the gaussian function based on the distance between the interpolated vertex position of the quad and the position of the RBF center.

Algorithm 10 Density injection - Main loop (CPU)

```
sort particles
nearest particle = first particle
farthest particle = first particle
push matrices and viewport properties
setup viewport and parallel projection
turn on FBO and shader
activate additive blending
z_lower_bound = LPV_left_lower_z
z_upper_bound = LPV_left_lower_z+slice_spacing
for s = 0 to nb_slices do
  // Update depth bounds
  bind slice s to FBO
  z_lower_bound += slice_spacing
  z_upper_bound += slice_spacing
  // Update iterators
  while nearest particle nearest border < z_upper_bound do
    nearest particle = next
  end while
  while farthest particle farthest border < z_lower_bound do
    farthest particle = next
  end while
  // Execute shader and evaluate RBFs
  render particle bounding quads from nearest particle to farthest particle
end for
deactivate additive blending
turn off FBO and shader
restore matrices and viewport properties
```

12.3.3 Geometry voxelization

If we want the medium and solid objects to mutually cast shadows on each other in a natural and unified manner, the ideal solution remains to be able to represent both entities under the same form.

As an example, in [EP90], Ebert and Parent worked with fragments whose densities were accumulated in an A-buffer. Solid surfaces were natively adaptated to the production of fragments (intersection between a view ray and a surface), and ordering then by distance from the camera as a preparation for scan line rendering was not a problem. To also insert volumes in the A-buffer, which is not natural, a series of volume fragments were generated along the view ray between each pair of surface fragments and inserted in the same list. All entities could then be illuminated and visualized in an unique rendering pass.

We were inspired by this kind of strategy, with the difference that volumes being naturally adaptated to our propagation scheme, we had to transpose solid surfaces into voxels, the common representation of all entities in our method. We implemented a "brute force" voxelization on

GPU, which is currently not optimized enough to reach real-time, we therefore have no choice but performing this step as a precompution. Please note that any existing method taking an OpenGL scene as input and able to generate a grid of voxels can be used in this pipeline, instead of our method.

We basically proceed as follows :

1. The scene is provided as a simple pointer to a function drawing all geometry.
2. Three temporary voxelization volumes are allocated, one for each X, Y or Z axis, with the same size as the extinction volume.
3. An orthogonal camera is positioned in front of the first X-slice of the extinction volume, aligned with the X axis.
4. The viewport is resized to match the dimensions of a slice of the extinction volume, and rendering is redirected to the first Z-slice of the X-volume.
5. We iteratively render each "slice" of the geometry in the corresponding slice of the X-volume, the output color being the normal vector of the surface.
6. We repeat the process with both the Y-volume and the Z-volume, so that surfaces aligned with one of the axes are at least rendered in the two other volumes.
7. The content of the X and Y volumes is rotated to rectify their orientation, and the three volumes are averaged before being merged with the medium into the extinction volume.

For convenience, normal vectors from surfaces are kept separated from medium voxels. All volumes being implemented as floating point $RGBA$ textures, each EV voxel can store four values : medium densities are stored in the first R component, and normal vectors are stores in the remaining GBA slots.

12.3.4 Occlusion propagation

12.3.4.1 The occlusion volume

Now that both the medium's density and the geometry were injected into the 3D grid, the first idea would be to propagate the radiance emitted from each point light throughout the portion of the scene covered by the grid, like [Kap09].

When propagating radiance from cell to cell, it is actually difficult to simulate the quadratic attenuation. Equally distributing radiance to all six neighbours make it dissipate too rapidly near the source, whereas undervaluing the attenuation brings additional energy at each step.

To propagate indirect lighting from surfaces, [Kap09] chooses to represent the radiance emitted from each LPV cell using two bands of spherical harmonics. This method has the advantage that four coefficients are enough to model the radiance outgoing towards all six direct neighbours, while keeping the directionality of the propagation, a much needed property for indirect radiance, where a reflecting surface cannot be considered as a simple point light.

However, to separate the light outgoing in a particular direction from the whole radiance emitted by the cell, an additional filtering step is required and, most important, this kind of

representation is very coarse and does not offer an accurate control of the way light is propagated and attenuated in straight line.

Even when no participating medium exists in the scene, problems described above may occur. Direct radiance from a point light source would decrease exponentially, and thus barely reach the borders of the propagation volume. Trying to decrease the attenuation at each step causes a gain in the overall outgoing radiance compared to the radiance previously received by the cell. Therefore, spherical harmonics does not seem a good solution when propagating direct illumination.

12.3.4.2 Our method

Our idea is the following :

- For the attenuation caused by the distance from the source, and since we know the exact position of each point light source, we can consider the euclidian distance of the source to the center of each cell, rather than the distance in the 6-connexity propagation path (In 6-connexity a cell shares each of its six square faces with a different neighbour).
- The only data which is contained in the cells and which we need to propagate using the 6-connexity neighbouring scheme is the information concerning the occlusion.
- We would like the radiance received by a particular cell to be affected only by occluding objects encountered on the direct straight path from the source to the center of the cell.

We propose this solution :

- We do not propagate the radiance, but rather the occlusions by integrating the medium's density, to obtain the optical depth $\tau(X, S)$ between each cell X and each light source.
- Each cell gathers occlusion from all six neighbours, and weights values based on both the real lighting direction $\vec{SX}$, and the orthogonal cell-to-cell direction.
- The quadratic radiance attenuation will be performed directly when rendering the medium.

At each step, since a scattering method is not convenient for the GPU, each cell instead gathers the incoming light from all six directions. The information about the optical depth depends on the position of the source, the propagation therefore has to be performed separately for each light However, since we know precisely the theoretical lighting direction, there is no need to distinguish the six portions of each cell's density outgoing in the six respective directions. For each light source, one value only is sufficient.

When gathering occlusion, the idea is that although each cell X receives optical depth from six directions, either orthogonal or opposed, we know that it actually comes from a single point source S, so that there is theoretically only one exact incoming direction $\vec{SX}$. We have to compute an average incoming density, where the six weights are determined by the similarity between the theoretical propagation direction $\vec{SX}$ and the 6-connexity cell-to-cell incoming direction, using a simple dot product. As the light cannot arrive from a direction opposed to source, optical depth incoming with a negative scalar product has to be discarded. Although optical depth reaching a cell for the first time will always come from the direction of the source, since we do not keep

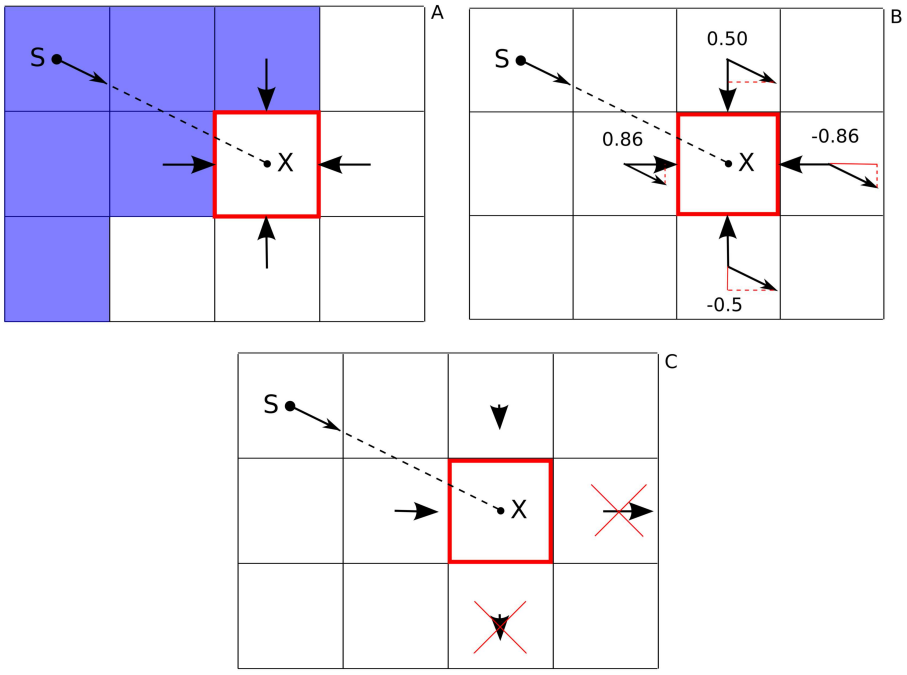

Figure 12.3: Occlusion gathering : weighting incoming directions. A : Cells already visited by the propagation wavefront are shows in blue, S is the position of the light source and X is the center of the cell for which the gathering process is detailed. B : We compute the dot product between the theoretical lighting direction $\vec{SX}$ and each orthogonal cell-to-cell direction. C : Neighbours whose dot product is negative are discarded.

any information about the incoming direction, this test is then necessary to prevent propagating backwards.

In other words :

$$\tau^{n+1}(X,S) = \frac{\sum_{i=0}^{5} W_{S,i}(X) T^n(X - \vec{d_i}, S)}{\sum_{i=0}^{5} W_{S,i}(X)} \tag{12.8}$$

where $\tau^n(X,S)$ is the portion of the density received from source S by cell X at step n, $\vec{d_i}$ is the incoming density direction from the i^{th} neighbouring cell, and which $W_{S,i}$ is the weight in the sum, given by :

$$W_{S,i} = (1 - K_t(X)) \max\left[\langle \vec{SX}, \vec{d_i} \rangle, 0\right] \tag{12.9}$$

12.3.4.3 Algorithm

The propagation algorithm has some similarities with the extinction injection.

Like to the extinction volume, the occlusion volume is stored as a 16-bit floating point texture. However, we need to store, for each cell, one value per light source, i.e. one texture per group of four lights. Since, both reading and writing in a texture is not available, the propagation process which

Algorithm 11 Propagation - Main loop (CPU)

```
push matrices and viewport properties
setup viewport and parallel projection
turn on FBO and shader
// — Initial propagation step — //
bind initial texture as read sampler
bind texture_1 as render target
execute shader on fullscreen quad
// — Main loop — //
for id_step=0 to nb_steps-1 step 2 do
  // Even steps
  bind texture_1 as read sampler
  bind texture_2 as render target
  execute shader on fullscreen quad
  // Odd steps
  bind texture_2 as read sampler
  bind texture_1 as render target
  execute shader on fullscreen quad
end for
turn off FBO and shader
restore matrices and viewport properties
```

is implemented on the GPU requires at least two copies of each texture. They are alternatively used either for reading or writing at each new step, where the previous result is overwritten.

Since the integration originates from a particular cell and propagates radially like a wavefront, the final occlusion value for each cell is obtained as soon as it is reached by the process. So, at each step, only the cells on the wavefront need to be considered. By marking the visited texels using an additionnal texture, we can speed up the process in the shader by discarding cells which either have already been computed, or have not been reached yet. Each flag texel must inform on whether each different light source has or not propagated its optical depth all way long to the respective cell. Indeed, one flag is needed per light and per occlusion grid cell.

To initialize each new propagation process, the two textures of the occlusion volume must be cleared so that each cell starts with zero, except for the ones on which lights sources were injected, which are initialized with their respective extinction coefficients. The two flags textures must also be cleared, except for the source's cells that start marked as already visited, by their respective source only.

As we notice that we need to keep the initial radiance (and flags) distribution to initialize each new propagation process, we can also store it in a third texture which would serve as the first step's read texture, and would be kept unchanged.

Each single propagation step involves a plane sweep along the texture slices. We switch to a parallel projection and resize the viewport to match the LPV's width and height. Most of the computation is performed in a fragment shader, called on a fullscreen quad to fill each slice.

Algorithm 11 shows the pseudo-code for the propagation main loop performed on the CPU. Algorithm 12 shows the pseudo-code of the fragment shader for the gathering process.

Algorithm 12 Propagation - Pixel shader (GPU)

$K(X)$ = Read local density from EV
for light source S **do**
  /* 1. Gather occlusions from neighbours */
  wghtd_dens_sum = 0
  weights_sum = 0
  for gathering direction $\vec{d_i}$ **do**
    $W_{S,i} = \max(\text{dot}(S\vec{X}, \vec{d_i}), 0)$
    $K(X - \vec{d_i})$ = Read neighbour density
    wghtd_dens_sum += $W_{S,i} * K(X - \vec{d_i})$
    weights_sum += $W_{S,i}$
  end for
  /* 2. Result for light S */
  pxl_channels[S] = $K(X)$ + wghtd_dens_sum / weights_sum
  pixel_color = pxl_channels
end for

12.3.5 Rendering

12.3.5.1 Visualizing the medium

Now that the information about occlusion has been propagated from each light through the volume, the final step is the visualization of both the surfaces and the medium. We need to solve equation 12.6, defining how to obtain the final color for each pixel.

Considering our model, we need to perform an integration between the observer and the nearest solid object. We have little choice but solving this integration using a conventional ray-marching over our grid.

This integration uses a fixed integration step, even if more sophisticated techniques can be used (see [GB10]).

The entire algorithm is implemented on a fragment shader. The ray-marching is performed separately for each light source. We start the integration from the observer and move in the direction of the surface in front of the camera.

Based on precomputations performed in the two previous sections, and considering equation 12.6, each step is straightforward :

1. Fetch $\rho(X)$, the extinction coefficient corresponding to the local density of our scattering medium, and accumulate it with the values fetched at previous steps. At this point, the density of the medium decides how much light will locally not pass through and thus be reflected, making the medium visible to the camera.

2. Fetch $\tau(X, S)$, the optical depth between X and each separate light source S. The radiance from S reaching X is obtained with : $e^{-\tau(X,S)}J(S)$.

3. Compute quadratic attenuation : $\|S - X\|^{-2}$.

4. Compute this step's contribution, and add the radiance in the integral result :

$$L_S(X) = (1 - K_t(X))L_S(X) + K_t L_{\text{IN}}(X) \tag{12.10}$$

where $L_{\text{IN}}(X)$ is the incoming radiance from the source, given by :

$$L_{\text{IN}}(X) = J(S) * e^{-\tau(X,S)} \tag{12.11}$$

Algorithm 13 shows the pseudo-code for the ray-marching using a pixel shader.

12.3.5.2 Rendering surfaces

Shading surfaces Last but not least, in order to render the surface in front of the current pixel, we have to use a simple and fast illumination model. In our implementation, we compute a simple Phong illumination, but other models can be used as well.

The final pixel color is finally obtained as the result of an additional integration step, outside the main ray-marching loop :

$$L_S(O) = Ph_S(P, O)e^{-T(O,P)-\tau(P,S)}\|S - X\|^{-2} \tag{12.12}$$

where $Ph_S(P, O)$ is the Phong illumination for light S, at position P on the surface, and as perceived by observer O.

Before entering the whole pipeline, we generate a G-buffer (a 3D texture, where each slice stores the different properties of each fragment) in which we store all properties needed for rendering :

- Color and depth buffers
- Positions of the camera and the nearest object in the scene, needed to fetch data from extinction and occlusion volumes
- Material properties of the surfaces, needed for surface illumination

Problem of self-occluded surfaces and our solution There is an inherent issue arising when manipulating both flat surfaces and their voxelized version. Surfaces are naturally shadowed by their own voxels, inside which, by definition, they are situated.

To address this problem, we are first compelled to set an unavoidable constraint on our voxels, stating that all objects must be *well-formed*, i.e. featuring a clear interior and exterior. At the propagation step, based on the normal to each surface, which was stored within each voxel, light is still blocked by the geometry, but with a slight offset in the grid, so that surfaces which lie inside their own voxels but which should normally receive light from the source can be illuminated.

Typically, when light arrives on a voxelized surface strongly oriented towards the source, light-blocking voxels become those situated behind the surface. On the contrary, when a surface turns its back to the source (the ray and the normal are oriented in the same direction), we assume that this surface must be shadowed since it may be situated at the rear of an object.

Algorithm 13 Medium rendering - Pixel shader (GPU)

```
set pixel_color to black
fetch camera_pos and object_pos from G-buffer
fetch object_depth from G-buffer
texcoord_offset=grid_pos-(grid_size * cell_size)/2
texcoord_factor=1/(grid_size * cell_size)
// — Loop on lights — //
for id_light=0 to nb_lights do
  density_accumulator=0
  set radiance_accumulator to black
  nb_steps=object_depth/step_length
  // — Loop on ray-marching steps — //
  for id_step=0 to nb_steps do
    pos=camera_pos+id_step*view_dir
    texcoord=
    ↪(pos-texcoord_offset)*texcoord_factor
    fetch cell_density from ext. vol. at texcoord
    density_accumulator+=cell_density
    fetch occlusion from occ. vol. at texcoord
    attenuation=1/pow(dist(pos,light_position),2)
    radiance_accumulator+=
    ↪cell_density*light_color*attenuation*
    ↪exp(-density_accumulator-occlusion)
  end for
  compute phong_illumination
  radiance_accumulator+= exp(-density_accumulator-occlusion)*
  ↪attenuation*phong_illumination
  pixel_color+=radiance_accumulator
end for
```

12.3.6 Using different resolutions for extinction and occlusion volumes

When visualizing participating media, the realism of the appearance of the scattering medium viewed directly seems more important visually than occlusions effects, which are, in current methods, always approximated with lower frequencies.

Since the propagation step appears costly, a simple trick that we implemented consists in decreasing the resolution of the occlusion volume, while possibly increasing the resolution of the extinction volume.

- At phases 1 and 2 (see 12.3.2), the medium's extinction function is sampled at a higher resolution, which is also used for the scene voxelization. However, this brings very few additional cost.

- At phase 3 (see section 12.3.4), although occlusion effects are downsampled, the propagation

Vol. resolution	1 light	2 lights	4 lights
24^3	166	87	42
32^3	111	58	29
40^3	78	40	19
64^3	35	17	8
80^3	23	11	5

Table 12.1: FPS results with a static scene (no voxelization), dynamic lighting and participating medium. Image resolution is 1024×768 pixels.

process is exponentially much faster (fewer propagation steps, fewer slices and pixels to shade at each step).

- At phase 4 (see section 12.3.5), the ray-marching is a bit more costly (more steps), but the visual appearance is of higher quality.

Different volume resolutions allow tuning the process more finely to fit the type of participating media being rendered.

12.4 Results and discussion

This algorithm has been implemented using GLSL, an Intel Core 2 Quad 2.8Ghz processor and a NVidia GeForce GTX 570 graphics card. Screen resolution is 1024×768.

12.4.1 Performance

There are a lot of independent parameters in our method which can have an impact on both performances and quality. Which computation phases are or are not performed at each new frame is the parameter which impacts most on the speed at runtime. Although classic ray-tracing based methods have to perform again the major part of the computations at each frame, our pipeline is very modular.

- Dynamic scene : perform the entire pipeline at each frame.
- Static scene, dynamic lighting : phase 2 (voxelization) can be precomputed.
- Static scene and lighting : unfortunately, in this case, if the medium is changing, phase 3 (occlusion propagation) is still necessary.
- Static scene and participating medium : phases 1 and 2 can be precomputed. If the lighting is dynamic, data about occlusions must be updated.
- Static scene, lighting and participating medium : phases 1 (density injection), 2 (voxelization) and 3 (occlusion propagation) can be precomputed once and for all when initializing the application.

Nb RBF	1	10	50	100	500	1000	5000	10000
FPS	111	110	110	106	98	86	42	11

Table 12.2: Performance comparison when changing the number of particles to inject in the extinction volume. Image resolution is 1024×768 pixels. Occlusion propagation volume is 32^3 voxels, with one light source.

Volumes size	All dynamic	Dynamic light & medium	Rendering only
24^3	13.4	156	277
32^3	9.9	106	227
40^3	8	76	192
64^3	4.8	34	135
80^3	3.6	24	111

Table 12.3: Comparing the cost of the same scene in three conditions. We use one light source and one large RBF, in our Cornell box. Image resolution is 1024×768 pixels.

Table 12.1 shows how the number of lights affects performances. With small volumes, adding n more light divides the speed by more than n. When the size of the medium increases, this ratio diminishes even more.

Table 12.2 shows in which extent the number of particles modeling the medium can weight on the entire pipeline. We notice that the more we add particles, the cheaper each particle is. Unlike the resolution of the volumes which actually affect all phases of the pipeline, the medium's dimensions almost only affect the density injection phase (and the voxelization which we precompute). For each frame, after phase 1, all informations about the shape of the medium is sampled then stored in the extinction volume, therefore the complexity of the medium does not depend on the number of RBFs anymore, but on the number of samples, given by the number of cells in the extinction volume.

Table 12.3 compares FPS results for a dynamic scene, a scene with static geometry and dynamic lighting, and a full static scene where only the last rendering step is performed. The scene consists in a single large gaussian particle in a simple Cornell box with a single light source. We can see that our cheap voxelization method remains interactive for moderate volumes, but we plan to replace it by an existing one which would be more efficient. For the rest, the results are satisfying, since for a scene with a dynamic medium and lighting, we almost stay real-time even with large volumes.

12.4.2 Visual results

There are three main parameters, which can affect the quality of the results :

- The resolution of the extinction volume, which will decide the frequency of the sampling of the extinction function, at the density injection phase (see section 12.3.2). To render a simple animated smoke, for example, a 20^3 extinction volume is sufficient, and a higher resolution will bring even better results. Below 15^3, the particles will slightly twinkle.

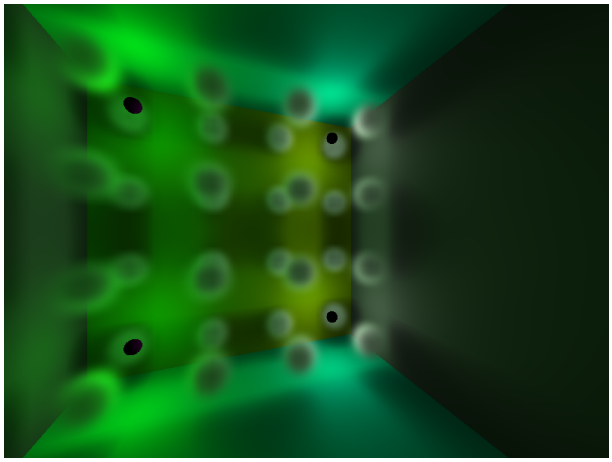

Figure 12.4: Several particles illuminated by four sources, which positions are shown by the small black spheres.

- The resolution of the occlusion volume, which will affect the quality of shadows and scattering effects. With a coarse extinction volume and a much finer occlusion volume (e.g. 10^3 and 30^3), despite good shadowing and scattering, the particles will look like cubes (i.e. grid cells). On the contrary, as explained in 12.3.6, a coarse occlusion volume and a large extinction volume will give a good direct lighting appearance to the particles, but raw shadows.

- The length of the ray-marching steps, which impacts on the quality of the radiance integration sampling (see 12.6). To avoid twinkling effects, it must not be higher than the size of both volumes cells.

Figure 12.4 illustrates shadowing effects by multiple particles.

Figure 12.5 illustrates multiple light scattering in a thick fog, with eight sources (four sources are behind the camera). The scattered radiance affects the appearance of the teapot.

Figure 12.6 illustrates glow effects, caused by single-scattering through a large fog filling the room.

Figure 12.9 shows the precision of our propagated shadows, with the thin hole in the teapot's handle accurately projected on the wall.

12.5 Conclusion

In this chapter, we presented a new method for illuminating and rendering heterogeneous isotropic scattering media in real time. The medium is modeled as a set of radial basis functions, which have proven a convenient representation for phenomena such as fog of smoke. The medium is first modeled by the user by providing a simple list of unarranged, heterogeneous particles of different amplitude and density.

Based on the idea behind Crytek's Lights Propagation Volumes [Kap09] for indirect lighting of surfaces, we solve the single scattering equation (see equation 12.6) by introducing a four-steps pipeline.

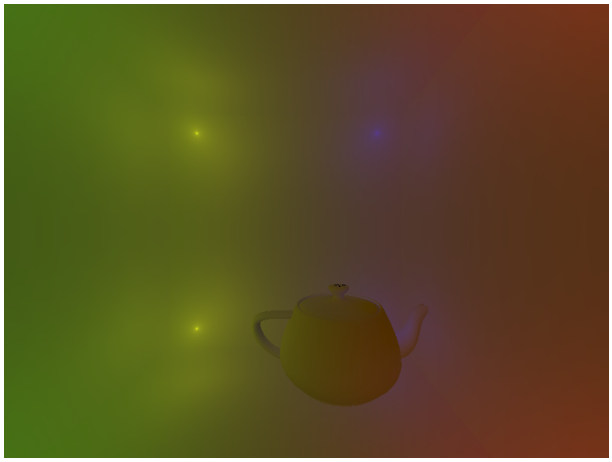

Figure 12.5: Multiple light scattering in a thick fog. OPV size is 50^3, image resolution is 1024×768.

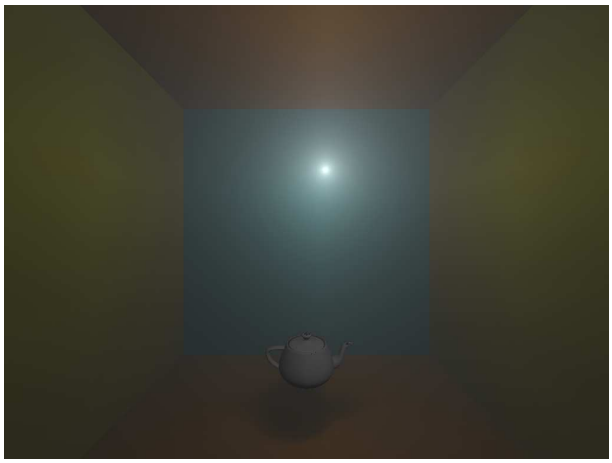

Figure 12.6: Glow around a point light source in a light fog. OPV size is 50^3, image resolution is 1024×768.

1. The extinction function of the medium is first injected into a volumetric grid, called the extinction volume, using a plane sweep along the grid's slices.

2. The geometry of the scene is voxelized, then injected in the extinction volume.

3. In a second grid, we compute the optical depth $\tau(X, S)$ required to evaluate the integral in the scattering equation, by propagating occlusions between each light source and each cell of the volume.

4. We finally render the medium and surfaces by performing a ray-marching between the camera and the nearest surface.

The optical depth $\tau(O, X)$ between the camera O and the current position X is fetched from the extinction volume, as the result of step 1, and accumulated at each new step.

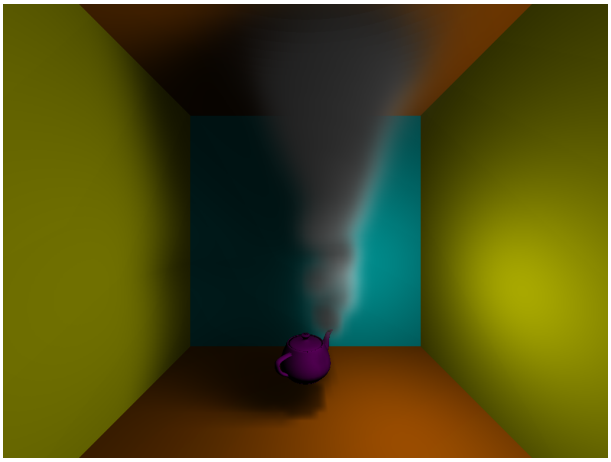

Figure 12.7: Smoke coming out of a teapot, both casting shadows. OPV size is 50^3, image resolution is 1024×768.

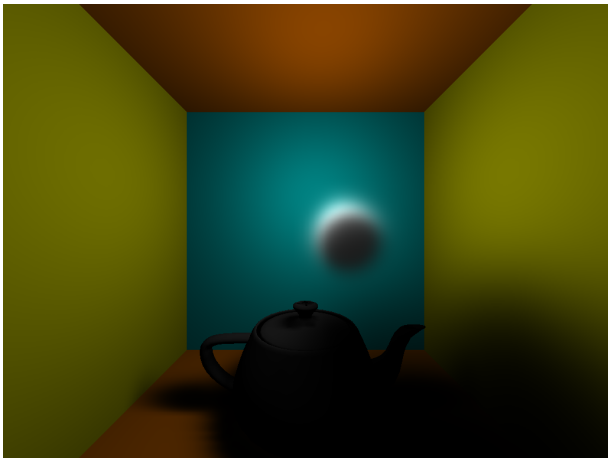

Figure 12.8: Occlusion on the teapot by the participating medium. OPV size is 50^3, image resolution is 1024×768.

The optical depth $\tau(X, S)$ between the position X and light source S is fetched from the occlusion volume, as the result of step 3.

When the ray-marching reaches the surface, it is illuminated based on the shadowing informations propagated throughout the occlusion volume.

Our method nevertheless has a few limitations, such as the speed of our geometry voxelization, which is very slow. One of the most important limitations is the cost of the 3D grids in memory. We managed to use volumes up to 200×200, with which be barely retain some interactivity, but we really need to work on a more optimized data structure to be enable larger resolutions.

Our method fully achieves real time on conventional hardware, and renders scattering effects such as halos around light sources within the medium with good quality. We handle occlusions by the geometry, while allowing the use of point light sources, whereas previous methods either only focused on the medium, used environment lighting, or techniques such as shadow mapping, where the light cannot lie within the medium. Using OpenGL and GLSL, we believe that it is easy to implement and most of all, easy to integrate in an existing graphics engine.

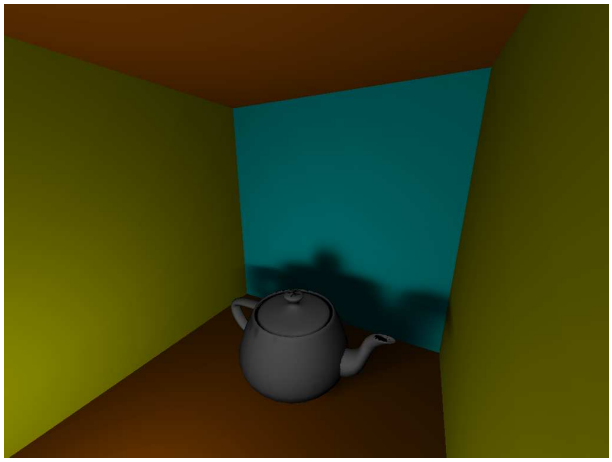

Figure 12.9: Shadow cast by the teapot and projected on the wall. OPV size is 80^3, image resolution is 1024×768.

However, in the very near future, we plan to bring several improvements to our method.

1. Handling more than eight light sources, even if this limitation is only related to our implementation in particular.
2. Optimizing the ray-marching, by only integrating along the portions of the view ray which actually bring a contribution.

We also plan to improve the scattering algorithm by decreasing the number of steps necessary to fill the volume, and optimize GPU memory cache management, in particular when fetching data from the volumes. We would also like to integrate our method in a wider pipeline, which would automatically generate sets of particles to design fog or smoke, and then render it with our method.

Chapter 13

Conclusion and perspectives

13.1 Summary

In this thesis, we dealt with the modelization, illumination and rendering of heterogeneous participating media in real-time. Real-time illumination is a very active area of computer graphics, especially since the advent of programmable graphics hardware, which continuously offer plenty of new possibilities. This is due to the exponential increase of the performances of GPUs, as well as a lot of new features at each new version of the most popular libraries like OpenGL or Direct3D. Within this domain, real-time rendering of heterogeneous participating media still poses a lot of challenges today. For rendering surfaces, there exist standardized methods implemented natively on GPUs in a pipeline of successive operations, which takes raw geometry data, computes the image using the rasterization and directly outputs the result on the screen. No such process exists for heterogeneous volumes.

After a quick theoretical refreshment on the physics of light in chapter 2, we established the equation of transfer in chapter 3, which expresses the radiance reaching the camera as function of all small quantities of light in-scattered along the view-ray.

In chapter 4, we recalled the basics of the wavelet transform. We started by defining how to build Haar, linear and quadratic bases of scaling functions and wavelets. We then addressed Mallat's fast wavelet transform in 1D and 2D.

In chapters 6 and 7, we reviewed the most notable of the existing works in the literature. While we reviewed ancient methods, we did compare the different ways of modeling analytical and heterogeneous media. We then discussed recent works, which we classified according to the type of phenomenon : outdoor fog, clouds and smoke. We concluded by presenting the choices we made for our methods.

In chapter 8, we presented a new method to model and render horizontal heterogeneous fog in real-time. We model the medium as a combination of a horizontal function basis, which allows a complete heterogeneity along the XZ plane, and a vertical extinction function, defined analytically in order to ease its integration. The distribution of the density is modeled in a Haar or b-spline basis very easily by loading a simple grayscale image containing the basis coefficients.

This is the first method to be based on a true heterogeneous fog designed using discrete samples, and rendered using the GPU. We are also the firsts to exploit wavelet compression to optimize the visualization process. As a precomputation step, the density distribution undergoes a wavelet decomposition, where the different frequencies of details are extracted in a series of summable

layers. At runtime, depending on the distance from the medium, we can dynamically choose which details are the most significant and worth being rendered.

We can quote two main advantages of using wavelets. First, the rendering algorithm being based on a grid traversal, its cost is closely related to the number of coefficients contributing along the view-ray. When extracting the details using wavelets, that number almost does not change, where a simple extraction of the details via an iterative downsampling on the grid of coefficients would only add more coefficients at each "decomposition" step. Second, the wavelet decomposition does not only works on the coefficients, but considers the final value obtained after the product of each coefficient with its basis function, yielding an exact result.

In chapter 9, we proposed two optimisations to this method. The first considers transposing the result of the wavelet decomposition into a single layer, where areas featuring low details have their resolution locally downgraded, giving rise to larger cells on the grid. The second method tries to go further, and corrects some of the drawbacks of the first optimization. This is a completely different strategy where the coefficients are considered individually, which allows dropping all areas with neglectable values, which nevertheless had to be traversed with the previous methods.

In chapter 10, we propose to generate density maps using fractal images. In comparison with other methods based on Perlin noise, fractal images represent a mathematical shape, on which we have a true control. By changing the parameter of the sets, we are able to easily obtain an interesting animation, coherent both in space and time.

In chapter 11, we started working on the illumination of our fog. We simulate occlusions by the geometry based on a shadow mapping, and also render shadows cast by the medium.

In chapter 12, we presented a new method to model and illuminate a participating medium using single-scattering from several light sources. We consider the idea behind light propagation volumes, a technique dedicated to the propagation of indirect light diffusely reflected on surfaces by a set of virtual point lights. Whereas the original LPVs scatter light isotropically in the air, we adapt it to the initial direct lighting step (before the first bounce), so that, in the absence of occluders along the way, light emitted from a point source can reach all unshadowed surfaces. We handle the presence of a participating medium in the scene, as well as all types of occlusions from the medium and the geometry, where the original LPVs did not consider occlusions from the geometry.

Our method is decomposed in four steps. The medium is first modeled as a set of particles, which are injected into a *density volume*. Then, the geometry is voxelized, and injected into the density volume as well. Third, the optical depth is iteratively propagated from each point source throughout an *occlusion propagation volume*. During this process, similarly to half-angle slicing, we do not uselessly integrate twice on the same cell. Finally, the medium is rendered based on a ray-marching along the view-ray.

13.2 Answering the problematics

In section 1.3, we listed some constraints and problematics that served as our guideline for this thesis. We believe to have addressed them :

- Can we find a novel, efficient and user-friendly way of modeling a natural phenomenon like fog, which would allow an optimized rendering on programmable GPUs ?

In chapter 8, we proposed a new method to render outdoor fog. The user can model the density distribution very easily under the form of a grayscale image, and load it in our implementation. The fog is initially modeled in a two-dimensional b-spline scaling functions basis, which are naturally smooth. It is therefore possible to design a spatially very large smooth fog covering an entire scene, only using the number of coefficients required by the resolution of its details.

- How can we dynamically accelerate the rendering of a heterogeneous medium ?

  This representation of the fog as a scaling functions basis enables a multiresolution analysis using the wavelet decomposition. The different layers of details that are generated are represented in memory using wavelet functions, which can, in this context, model the information of two coefficients with only one. As a result, the medium's density keeps almost the same size in memory before the decomposition and after having extracted the different resolutions of details.

- How can heterogeneous media be animated analytically, without the need for a complex physical simulation ?

  In chapter 10, we showed how it can be possible to generate chaotic fogs using fractals, and how such density distributions can be animated by simply modifying a single parameter. We obtain interesting animations, in which the whole density distribution changes at each frame, and where the smallest details are animated more rapidly than the lower frequencies, all this in a spatially coherent fashion.

- Can we find an efficient way of computing optical depth from sources through a volume in real-time using the GPU ?

  In chapter 12, we presented a method to illuminate a single-scattering medium like smoke, by propagating occlusions from the sources throughout a grid of voxels. Whereas computing optical depth from each step along the view ray would lead to redundant integrations along identical paths in the grid, this propagation scheme makes it possible to process each voxel only once.

- Can we account for occlusions by both solid geometry as well as a volume in an unified strategy ?

  In our method, the medium is provided as a set of blobs, and the scene is provided as a pointer towards a function drawing OpenGL geometry. The two first steps consist in transforming both of them in a set of voxels, which are then injected into a common *density volume*. Thanks to this common representation, we are able to indifferently simulate shadows by and onto both types of entities.

We are rather satisfied with the results obtained, even if we wish we had more time to implement some of the other methods, in order to be able to make deepeer side by side comparisons with our works, on identical hardware. Participating media is a true inspiring domain, and we would have appreciated to turn all our ideas into publications in time.

13.3 Future work

In the near future, we need to complete the illumination of our fog. For the moment, the shadow from the medium on the geometry is obtained by means of a basic projection from its horizontal plane, and we would like to truely account for the optical depth between the surface and the light source, like this is currently the case for the fog's self-shadowing. We need to improve the computation of the occlusions between a cell on the fog plane and the source, by trying to find the less computationally expensive way of accounting for all traversed cells. We would like to go further regarding the use of fractals to generate a density distribution, and see if it is possible to combine several sets with different parameters in order to generate small scale visual details. Also, we would like to work on more sophisticated criteria for deciding which details to keep or to discard, by trying to determine visually whether a coefficient brings any contribution or not, regardless of its distance from the camera. In addition, we would like to port our ray-marching algorithm on CUDA and see this could open a door to further improvements, such as replacing the ray-marching by a graph-marching.

We also have further projects regarding our occlusion propagation method, which are currently in development. We are working on a new method to simulate multiple diffuse bounces of light between surfaces and inside the volume. Because this involves more complex memory accesses, we are porting the propagation step on CUDA, instead of GLSL shaders. While the propagation of direct lighting remains almost unchanged, our idea is to speed-up subsequent bounces by propagating frustums of light from the illuminated surfaces and through the grid of voxels, in order to approximate a completely diffuse indirect illumination.

We also consider working on the simulation and rendering of clouds and atmospheric scattering, using CUDA to try, in real-time, simulating complex phase function effects such as the rainbow and which involve wavelength-dependent scattering and dispersion.

The end.

Publications

13.4 International conferences

- [GB10] *Modeling and Rendering Heterogeneous Fog in Real-Time Using B-Spline Wavelets*
  Anthony GIROUD, Venceslas BIRI
  Proceedings of WSCG'2010
  Plzeň, Czech Republic

- [GB11] *Illuminating and Rendering Heterogeneous Participating Media in Real-Time Using Opacity Propagation*
  Anthony GIROUD, Venceslas BIRI
  Proceedings of GRAPP'2011
  Algarve, Portugal

- [BG10] *Using Chaotic Maps for Heterogeneous Fog Rendering in Computer Graphics*
  Venceslas BIRI, Anthony GIROUD
  Proceedings of Chaos'2010
  Chania, Crete, Greece

13.5 Seminars

- Seminar at the University Of Keio, Yokohama, Japan
  August 17$^{\text{th}}$, 2009

13.6 Domestic conferences

- [GB09] *Modélisation et rendu temps-réel de brouillard hétérogène à l'aide d'ondelettes*
  Anthony GIROUD, Venceslas BIRI
  Proceedings of AFIG'2009
  Arles, France

Bibliography

[Arv93] James Arvo. Transfer equations in global illumination, 1993.

[BG10] V. Biri and A. Giroud. Using chaotic maps for heterogeneous fog rendering in computer graphics. In *Proceedings of Chaos'2010*, 2010.

[BGMR83] James Blinn, Julian Gomez, Nelson Max, and William Reeves. The simulation of natural phenomena (panel session). In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '83, pages 137–139, New York, NY, USA, 1983. ACM. Chairman-Csuri, Charles A.

[BHZK05] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. High-quality surface splatting on today's gpu's. In *Point-Based Graphics, 2005. Eurographics/IEEE VGTC Symposium Proceedings*, pages 17–141, 2005.

[Bli82a] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[Bli82b] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *SIGGRAPH '82: Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pages 21–29, New York, NY, USA, 1982. ACM Press.

[BMA02] Venceslas Biri, Sylvain Michelin, and Didier Arquès. Real-time animation of realistic fog. In *Proceedings of 13th Eurographics Workshop on Rendering, poster session*, pages 9–16, 2002.

[Bou08] Antoine Bouthors. *Realistic rendering of clouds in real-time / Rendu réaliste de nuages en temps-réel*. Phd thesis, Université Joseph Fourier, june 2008.

[Car84] Loren Carpenter. The a -buffer, an antialiased hidden surface method. In *SIGGRAPH '84 Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '84, pages 103–108, New York, NY, USA, 1984. ACM.

[CBDJ11] Jiawen Chen, Ilya Baran, Frédo Durand, and Wojciech Jarosz. Real-time volumetric shadows using 1d min-max mipmaps. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 39–46 PAGE@7, New York, NY, USA, 2011. ACM.

[CCWG88] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. In *SIGGRAPH '88 Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '88, pages 75–84, New York, NY, USA, 1988. ACM.

[CG85] Michael F. Cohen and Donald P. Greenberg. The hemi-cube: a radiosity solution for complex environments. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '85, pages 31–40, New York, NY, USA, 1985. ACM.

[Cha50] S. Chandrasekhar. *Radiative Transfer*. Dover, 1st edition, 1950.

[CPCP+05] Eva Cerezo, Frederic Perez-Cazorla, Xavier Pueyo, Francisco Seron, and François X. Sillion. A survey on participating media rendering techniques. *the Visual Computer*, 2005.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '77, pages 242–248, New York, NY, USA, 1977. ACM.

[CS92] William M. Cornette and Joseph G. Shanks. Physical reasonable analytic expression for the single-scattering phase function. *Applied optics, vol. 31, no. 6*, pages 3152–3160, 1992.

[CT82] Robert L. Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1):7–24, January 1982.

[DDJC95] Laurent Da Dalto, Jean-Pierre Jessel, and René Caubet. Fast rendering of participating media in a global illumination application. In *International Conference on Visualization and Modelling, Leeds*, pages 1–16. British Computer Society, december 1995.

[DKY+00] Yoshinori Dobashi, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, and Tomoyuki Nishita. A simple, efficient method for realistic animation of clouds. In *SIGGRAPH '00 Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 19–28, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[DNO98] Yoshinori Dobashi, Tomoyuki Nishita, and Tsuyoshi Okita. Animation of clouds using cellular automaton. In *Proceedings of Computer Graphics and Imaging '98*, pages 251–256, 1998.

[Dom06] Dariusz Domański. Fast algorithm for cloud rendering using flat "3d textures", 2006.

[DS05] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05, pages 203–231, New York, NY, USA, 2005. ACM.

[Duf85] Tom Duff. Compositing 3-d rendered images. In *SIGGRAPH '85 Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '85, pages 41–44, New York, NY, USA, 1985. ACM.

[DYN02] Yoshinori Dobashi, Tsuyoshi Yamamoto, and Tomoyuki Nishita. Interactive rendering of atmospheric scattering effects using graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '02, pages 99–107, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[EP90] David S. Ebert and Richard E. Parent. Rendering and animation of gaseous phenomena by combining fast volume and scanline A-buffer techniques. *SIGGRAPH Computer Graphics*, 24(4):357–366, September 1990.

[Fat09] Raanan Fattal. Participating media illumination using light propagation maps. *ACM Trans. Graph.*, 28(1):7:1–7:11, February 2009.

[FSJ01] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *SIGGRAPH '01 Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 15–22, New York, NY, USA, 2001. ACM.

[GB09] A. Giroud and V. Biri. Modélisation et rendu temps-réel de brouillard hétérogène à l'aide d'ondelettes. In *Proceedings of AFIG'2009*, 2009.

[GB10] A. Giroud and V. Biri. Modeling and Rendering Heterogeneous Fog in Real-Time Using B-Spline Wavelets. In *Proceedings of WSCG'2010*, pages 145–152, 2010.

[GB11] A. Giroud and V. Biri. Illuminating and Rendering Heterogeneous Participating Media in Real-Time Using Opacity Propagation. In *Proceedings of GRAPP'2011*, pages 113–118, 2011.

[GPC+12] A. Gruson, A.H. Patil, R. Cozot, K. Bouatouch, and S. Pattanaik. Light propagation maps on parallel graphics architectures. In *Eurographics Symposium on Parallel Graphics and Visualization*, page 81–88. The Eurographics Association, 2012.

[GTG84] Cindy M. Goral, Kenneth E. Torrance, and Donald P. Greenberg. Modeling the interaction of light between diffuse surfaces. In *Computer Graphics 18, 3*, pages 213–222, 1984.

[HS67] Hoyt C. Hottel and Adel F. Sarofim. *Radiative Transfer*. McGraw-Hill, New York, 1967.

[Ina89] Masa Inakage. An illumination model for atmospheric environment. In *Proceedings of CGI '89*, pages 533–548, 1989.

[Irw96] John Irwin. Full-spectral rendering of the Earth's atmosphere using a physical model of Rayleigh scattering. In *In Proceedings of the 14th Eurographics UK Conference*, pages 103–115, 1996.

[Kaj86] James T. Kajiya. The rendering equation. *SIGGRAPH Computer Graphics*, 20(4):143–150, August 1986.

[Kap09] Anton Kaplanyan. Advances in real-time rendering in 3d graphics and games course. In *SIGGRAPH 2009*, 2009.

[KKH02] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Multi-dimensional transfer functions for interactive volume rendering. *TVCG 8, 4 (July 2002)*, page 270–285, 2002.

[Kla87] R. Victor Klassen. Modeling the effect of the atmosphere on light. *ACM Transactions on Graphics, Volume 6*, pages 215–237, 1987.

[KONN91] Kazufumi Kaneda, Takashi Okamoto, Eihachiro Nakamae, and Tomoyuki Nishita. Photorealistic image synthesis for outdoor scenery under various atmospheric conditions. *The Visual Computer*, pages 247–258, 1991.

[KPH+03] Joe Kniss, Simon Premože, Charles Hansen, Peter Shirley, and Allen McPherson. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics*, pages 150–162, 2003.

[KVH84] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. In *SIGGRAPH '84 Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, 1984.

[LH06] Sylvain Lefebvre and Hugues Hoppe. Perfect spatial hashing. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 579–588, New York, NY, USA, 2006. ACM.

[LHCL05] Horng-Shyang Liao, Tan-Chi Ho, Jung-Hong Chuang, and Cheng-Chung Lin. Fast rendering of dynamic clouds. *Computer Graphics*, 29(1):29–40, February 2005.

[LMAK00] Pascal Lecocq, Sylvain Michelin, Didier Arquès, and Arthur Kemeny. Simulation d'éclairage en présence de milieux participatifs : vers une solution temps réel. In *Proceedings of AFIG'2000*, pages 147–156, 2000.

[Mal93] Tom Malzbender. Fourier volume rendering. *ACM Transactions on Graphics, Volume 12 Issue 3*, pages 233–250, 1993.

[Mal08] S. Mallat. *A Wavelet Tour Of Signal Procesing.* Academic Press, 3rd edition, 2008.

[Man06] Petr Man. Generating and real-time rendering of clouds. In *Proceedings of CESCG' 06*, 2006.

[Max86] Nelson L. Max. Atmospheric illumination and shadows. In *SIGGRAPH '86 Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 117–124, New York, NY, USA, 1986. ACM Press.

[MHLH05] Marcus. A. Magnor, Kristian Hildebrand, Andrei Lintu, and Andrew J. Hanson. Reflection nebula visualization. *IEEE Visualization 2005*, pages 255–262, 2005.

[NDN96] Tomoyuki Nishita, Yoshinori Dobashi, and Eihachiro Nakamae. Display of clouds taking into account multiple anisotropic scattering and skylight. In *SIGGRAPH '96 Proceedings of the 23th annual conference on Computer graphics and interactive techniques*, pages 379–386, June 1996.

[NMN87] Tomoyuki Nishita, Yasuhiro Miyawaki, and Eihachiro Nakamae. A shading model for atmospheric scattering considering luminous distribution of light sources. In *SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, volume 21(4), pages 303–310, 1987.

[NN86] Tomoyuki Nishita and Eihachiro Nakamae. Continuous tone representation of three-dimensional objects illuminated by sky light. *SIGGRAPH Computer Graphics*, 20(4):125–132, August 1986.

[NN94] Tomoyuki Nishita and Eihachiro Nakamae. Method of displaying optical effects within water using accumulation buffer. In *SIGGRAPH '94 Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 373–379, 1994.

[NR92] K. Nagel and E. Raschke. Self-organizing criticality in cloud formation ? *Physica A, 182*, pages 519–531, 1992.

[NSTN93] Tomoyuki Nishita, Takao Sirai, Katsumi Tadamura, and Eihachiro Nakamae. Display of the Earth taking into account atmospheric scattering. In *SIGGRAPH '93 Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 175–182, 1993.

[PAS03] Simon Premože, Michael Ashikhmin, and Peter Shirley. Path integration for light transport in volumes. In *Proceedings of Eurographics Symposium on Rendering '03*, pages 1–12, 2003.

[Per85] Ken Perlin. An image synthesizer. In *SIGGRAPH '85 Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '85, pages 287–296, New York, NY, USA, 1985. ACM.

[REK+04] Kirk Riley, David S. Ebert, Martin Kraus, Jerry Tessendorf, and Charles Hansen. Efficient rendering of atmospheric phenomena, 2004.

[RGK+08] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. In *ACM SIGGRAPH Asia 2008 papers*, SIGGRAPH Asia '08, pages 129:1–129:8, New York, NY, USA, 2008. ACM.

[RNGF03] Nick Rasmussen, Duc Quang Nguyen, Willi Geiger, and Ronald Fedkiw. Smoke simulation for large scale phenomena. *ACM Transactions on Graphics*, 22(3):703–707, July 2003.

[RT87] Holly Rushmeier and Kenneth Torrance. The zonal method for calculating light intensities in the presence of participating medium. In *SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, volume 21(4), pages 293–302, 1987.

[RZLG08] Zhong Ren, Kun Zhou, Stephen Lin, and Baining Guo. *Gradient-based Interpolation and Sampling for Real-time Rendering of Inhomogeneous, Single-scattering Media*. Technical report, Microsoft Research Asia, 2008.

[SDS95] E. J. Stollnitz, A. D. Derose, and D. H. Salesin. Wavelets for computer graphics: a primer.1. *Computer Graphics and Applications, IEEE*, 15(3):76–84, 1995.

[SF93] Jos Stam and Eugene Fiume. Turbulent wind fields for gaseous phenomena. In *SIGGRAPH '94 Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 369–376, 1993.

[SH92] Robert Siegel and John R. Howell. *Thermal Radiation Heat Transfert*. Hemisphere Publishing, 3rd edition, 1992.

[Slo08] P. Sloan. Stupid spherical harmonics (sh) tricks. In *GDC'08*, 2008.

[Vos83] Richard Voss. Fourier synthesis of Gaussian fractals : noises, landscapes and flakes. In *SIGGRAPH '83 Tutorial on state of the art image synthesis, v.10*, 1983.

[Wat90] Mark Watt. Light-water interaction using backward beam tracing. In *SIGGRAPH '90 Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 377–385, 1990.

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, June 1980.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, August 1978.

[ZCS07] Tianshu Zhou, Jim X. Chen, and Peter Smith. Real-time fog using post-processing in OpenGL. In *Proceedings of HCI' 2007*, pages 165–174, 2007.

[Zdr04] Dorota Zdrojewska. Real-time rendering of heterogeneous fog based on the graphics hardware acceleration. In *Proceedings of CESCG' 04*, 2004.

[ZHG+07a] K. Zhou, Q. Hou, M. Gong, J. Snyder, B. Guo, and H.-Y. Shum. Fogshop: Real-time design and rendering of inhomogeneous, single-scattering media. In *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 116–125. IEEE Computer Society, 2007.

[ZHG+07b] Kun Zhou, Qiming Hou, Minmin Gong, John Snyder, Baining Guo, and Heung-Yeung Shum. Fogshop : Real-time design and rendering of inhomogeneous, single-scattering media. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, PG '07, pages 116–125, Washington, DC, USA, 2007. IEEE Computer Society.

[ZRL+08] Kun Zhou, Zhong Ren, Stephen Lin, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Real-time smoke rendering using compensated ray marching. *ACM Transactions on Graphics*, 27(3):36:1–36:12, August 2008.